Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Debevec

# Razvoj spletne aplikacije za vzdrževanje podatkov o projektih z uporabo metodologije Scrum

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Viljan Mahnič

Ljubljana, 2017

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Razvoj spletne aplikacije za vzdrževanje podatkov o projektih z uporabo metodologije Scrum

Tematika naloge:

Za potrebe slovenskega podjetja, ki se ukvarja z razvojem informacijskih sistemov, izdelajte spletno aplikacijo, ki bo uporabnikom omogočala celovit pregled nad izvajanjem projektov. Pri razvoju uporabite metodologijo Scrum, zato v nalogi poleg podrobnega opisa aplikacije in njene realizacije posvetite ustrezno pozornost tudi predstavitvi metodologije. Podrobno opišite, kako je bila metodologija uporabljena na tem projektu ter analizirajte prednosti in pomanjkljivosti njene uporabe.

*Zahvaljujem se mentorju prof. dr. Viljanu Mahniču za odzivnost, pomoč in vodenje pri izdelavi diplomskega dela. Še posebaj pa se zahvaljujem družini in prijateljem za vso podporo v času študija. Zahvala gre tudi prijatelju Mitji Puglju, ki mi je omogočil in pomagal pri realizaciji diplomskega dela.*

# Kazalo

# Povzetek

**Naslov:** Razvoj spletne aplikacije za vzdrževanje podatkov o projektih z uporabo metodologije Scrum

**Avtor:** Nejc Debevec

V diplomskem delu je predstavljen razvoj spletne aplikacije za planiranje in spremljanje poslovne uspešnosti podjetja. Na začetku je opisana metodologija Scrum, katero smo uporabili za razvoj projekta. Opisali smo glavne sestavne dele metodologije, katere smo uporabljali pri razvoju. Sledi opis vseh uporabljenih tehnologij na projektu, tu smo pri vsaki tehnologiji predsavili pomembnejše lastnosti, katere tehnologija omogoči razvijalcem. Največji delež diplomske naloge je namenjen predstavitvi implementacije aplikacije. Tu opišemo vse komponente aplikacije in sicer kako izgledajo, kakšne funkcionalnosti ponujajo in pri nakaterih smo podrobnejše opisali njeno rešitev. Na koncu je predstavljen sam potek razvoja po iteracijah, torej opis vsake iteracije, kaj je bilo implementirano, s kakšnimi problemi smo se srečevali, ter kakšna je bila njena uspešnost. Narejena je tudi krašja analiza uporabe metodologije Scrum. Tu opišemo dobre in slabe stvari, posameznih delov metodologije in predstavimo ideje za izboljšanje našega izvajanja procesa Scrum.

**Ključne besede:** razvoj spletnih aplikacij, agilne metodologije, Scrum, vodenje projektov.

# Abstract

**Title:** Development of a web application for maintaining project data using Scrum

**Author:** Nejc Debevec

The diploma thesis presents the development of the web application designed for planning and monitoring business performance of a company. Initially, the work describes the Scrum methodology and its main components, which were used as the basis for the development of the project, followed by a description of all the technologies used in this project. At this point, the most important features that these technologies have to offer to developers are presented. The diploma thesis mostly focuses on presenting the implementation of the application. Here all the components of the application are described, namely their appearance, information on what functionality they can offer and, with some, even more detailed descriptions of their production. Finally, the process of development of individual iterations is presented. Every iteration is described; what was implemented into it, what problems were encountered and what its success was. A short analysis of the use of the Scrum methodology was also made. Here, the positive and negative characteristics of the individual parts of the methodology are given, and new ideas to improve the implementation of the Scrum process itself are also presented.

**Keywords:** web application development, agile methodologies, Scrum, project management.

# Poglavje 1

# Uvod

Vsako podjetje, ki želi biti uspešno in ima željo po rasti, tako finančno kot kadrovsko mora imeti zastavljeno vizijo, ki jih bo vodila do uspehov. Vsaka vizija je sestavljena iz več manjših ciljev, ki vodijo do vizije. Za vsako podjetje je dobro, da ima za projekte zastavljene cilje. Ker živimo v obdobju tehnologije, je potreba po spletnih aplikacijah vse večja, zato je smiselno, da se evidenca zastavljenih ciljev/planov prenese na spletno aplikacijo.

Avtor diplomskega dela je zaposlen kot študent pri podjetju, ki se ukvarja z implementacijo informacijskih sistemov in razvojem tehnoloških rešitev po meri. Dobil je nalogo, da razvije spletno aplikacijo, ki podjetju omogoča planiranje finančnih prihodkov in odhodkov projektov. Na podlagi planiranih finančnih podatkov naredimo analizo podatkov, primerjavo planiranih planov z realiziranimi in uporabniku z interaktivnimi grafi prikažemo uspešnost planiranja.

Najpopularnejša metoda za razvoj programske opreme je Scrum [19, 13, 12]. Popularnost temelji na tem, da podpira vmesne spremembe zahtev, prilagajanje razmeram na trgu. Namenjena je dinamičnemu razvoju programske opreme. Ker vse zahteve še niso bile točno definirane, smo se za metodo razvoja programske opreme odločili za Scrum. Z metodo Scrum so v podjetju že dodobra spoznani, saj ga aktivno uporabljajo že nekaj let. Sami še nismo imeli izkušenj z razvojem programske opreme po metodi Scrum, zato smo se

odločili, da se z razvojem te spletne aplikacije vpeljemo tudi v metodologijo Scrum.

V diplomskem delu je v začetnih poglavjih opisana metodologija Scrum, njene glavne značilnosti in potek procesa dela. Sledi poglavje o tehnologijah, tu smo opisali glavne tehnologije, ki so bile uporabljene pri razvoju spletne aplikacije. Nato predstavimo razvoj spletne aplikacije. Na začetku spletno aplikacijo podrobneje opišemo, nato pa prikažemo razvoj po posameznih pogledih v aplikaciji. Poleg pogledov opišemo tudi vse dodatne funkcionalnosti, ki so bile implementirane. V zadnjem poglavju opišemo, kako je bila uporabljena metodologija Scruma pri razvoju aplikacije. Najprej predstavimo vloge, ki so bile dodeljenje, nato opišemo kako smo določili, da bo potekalo delo, sledi opis implementacije zahtev po posameznih iteracijah in na koncu poglavja predstavimo analizo uporabe Scruma pri razvoju naše spletne aplikacije.

# Poglavje 2

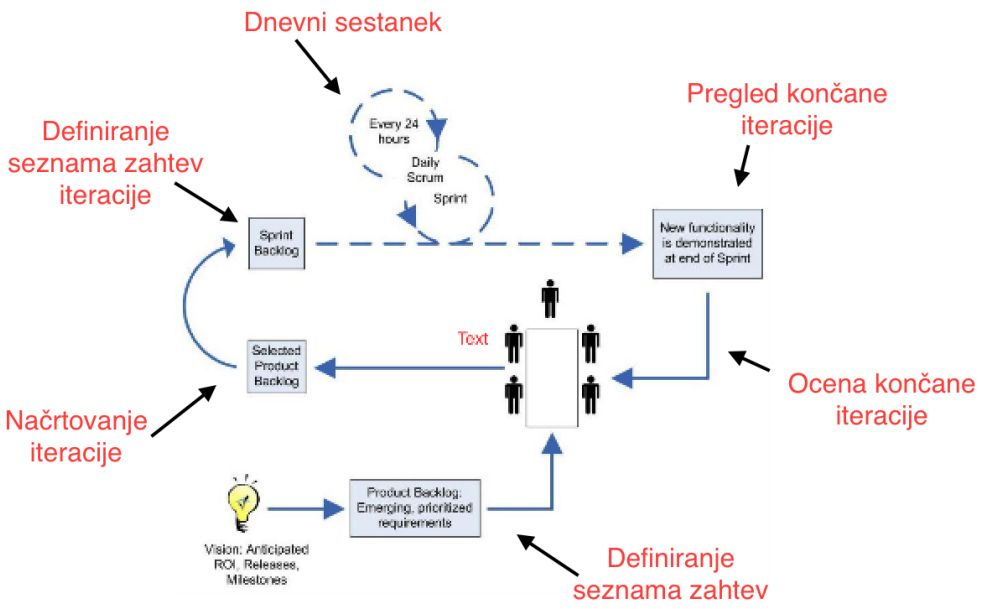
# SCRUM

Agilne metodologije opisujejo množico načel programskega razvoja, za katerega se zahteve in rešitve spreminjajo skozi razvoj. Poudarek pri agilnem razvoju je na ekipnem delu, pogostem objavljanju delujočih delov produkta, tesno sodelovanje z naročnikom ter hiter in fleksibilen odziv na spremembo. Zaradi podpore le-teh se agilne metodologije držijo načela, naj razvojna ekipa vsebuje nadpovprečno sposobne in izkušene razvijalce, saj se ti lažje prilagajajo vmesnim spremembam.

## 2.1 Opis metodologije Scrum

Scrum je najbolj razširjena in uporabljena agilna metoda, saj je preprosta in enostavna za implementacijo. Poudarek te metode je na ustvarjalnosti, prožnosti in prilagodljivosti. Lastnost metodologije Scrum je, da je interaktiven in inkrementalen proces. To pomeni, da razvoj poteka v iteracijah (ang. Sprint), ki si sledijo ena za drugo. Po vsaki končani iteraciji dobi izdelek novo funkcionalnost. Cilje projekta določi naročnik, organizacija dela razvojne skupine za dosego teh ciljev je odvisna od same razvojne skupine. Vsebina tega poglavja je povzeta iz naslednjih virov [19, 13, 12]. Slika 2.1 prikazuje proces metodologije Scrum.

Jedro metodologije Scrum sestavljajo tri vloge: skrbnik metodologije




Slika 2.1: Proces metodologije Scrum [19].

(ang. Scrum master), skrbnik izdelka (ang. Product Owner), razvojna skupina (ang. Development team).

## 2.2 Vloge v metodologiji Scrum

Scrum predpisuje naslednje tri vloge skrbnik izdelka, skrbnik metodologije in razvojna skupina.

### 2.2.1 Skrbnik izdelka

Glavna naloga skrbnika izdelka (ang. Product Owner) je, da zastopa naročnika in interese vseh uporabnikov. Ker je skrbnik odgovoren, da je projekt narejen, kot je bil načrtovan oziroma naročen, mora imeti vizijo. Na skrbniku izdelka je, da to vizijo posreduje razvojni skupini. To naredi preko seznama

zahtev, ki vsebuje uporabniške zgodbe, ki jih je prioritiziral. To, da je vizija projekta pravilno posredovana, je ključnega pomena. Zaradi dobrih praks je pogosto naloga skrbnika izdelka dodeljena nekomu, ki dobro razume trg, konkurenco, uporabnike in je seznanjen s trenutnimi trendi, ki so pomembni za razvoj projekta. Poleg tega mora biti skrbnik komunikativen in razpoložljiv.

Zelo pomembno je, da je skrbnik vedno na voljo razvojni skupini za pomoč pri razumevanju zahtev oziroma celotnega projekta. Skrbnik nima nobene besede glede tega, na kakšen način oziroma v kolikšnih interakcijah se bo projekt implementiral. Poleg vseh omenjenih nalog skrbnik skrbi tudi zato, da je razvojna skupina primerno motivirana za delo.

### 2.2.2 Skrbnik metodologije

Proces Scrum je vzdrževan s strani skrbnika metodologije (ang. Scrum Master). Njegova naloga je, da so vsi vpleteni seznanjeni z metodologijo in da se proces dosledno izvaja na vseh ravneh organizacije. Pri vsaki iteraciji (ang. Sprint), nadzira razdelitev dela, prepreči obremenitve posameznega razvijalca in s tem zagotovi, da je iteracija realno planirana. Torej poskrbi zato, da na koncu iteracije ne pride do ne zaključenih nalog.

Skrbnik metodologije mora poskrbeti, da metodologija daje želene rezultate, v primeru problemov, da ugotovi, kaj je narobe in to popravi. Razvojno skupino varuje pred nepotrebnimi ovirami, tako poskrbi za nemoten procesni razvoj in pripomore k večji produktivnosti le-te. Ena izmed glavnih odgovornosti skrbnika metodologije je ta, da je seznam zahtev jasno predstavljen razvojni skupini, tako da ne pride do napak pri razvoju produkta.

### 2.2.3 Razvojna skupina

Razvojna skupina (ang. Development Team) je zadolžena za implementacijo zahtev iz seznama v posameznih iteracijah. Metodologija predpisuje, da naj bo skupina velika od tri do devet razvijalcev. Razvojna skupina je sestavljena tako, da pokrije vse tehnološke zahteve projekta, kar prepreči nepotrebne

težave pri razvoju produkta.

Najpomembnejša lastnost vsakega člana je sposobnost "ekipnega dela", torej da je posamezen član zmožen sodelovati z ostalimi. Vsak član razvojne skupine se sam odloči, koliko zahtev bo lahko opravil v določeni iteraciji, vendar je potem odgovornost na njem, da se te zahteve res zaključijo. Za razvoj je odgovorna celotna razvojna skupina, tako, da se v praksi nikoli ne izpostavljajo posamezni člani.

## 2.3 Seznam zahtev in uporabniške zgodbe

Seznam zahtev vsebuje uporabniške zgodbe, ki jih definira skrbnik izdelka. Uporabniške zgodbe so pravzaprav krajši opisi zahtev funkcionalnosti, za katere naročnik želi, da jih njegov produkt vsebuje. Opis uporabniške zgodbe mora biti kratek in jasen, da razvijalec nima težav z razumevanjem zahteve in lahko realno oceni število delovnih enot, potrebnih za implementacijo le-teh.

Seznam zahtev je sestavljen iz: funkcionalnih zahtev, popravkov in nefunkcionalnih zahtev. Funkcionalne zahteve se navezujejo na naročnikove želje in trende na trgu. Popravki so že razvite funkcionalne zahteve, vendar niso bile pravilno razvite oziroma so pomanjkljive. Nefunkcionalne zahteve se navezujejo na zahteve, ki so pomembne za razvoj, vendar nimajo funkcionalnih koristi za produkt. Glavna značilnost seznama zahtev v agilnih metodologijah je ta, da seznam ni nikoli dokončen, saj se med razvojem dopolnjuje in spreminja glede na naročnikove želje oziroma glede na trenutne razmere na trgu.

## 2.4 Hitrost razvoja

Hitrost razvoja nam pove, koliko dela je sposobna razvojna skupina opraviti v določeni iteraciji. Meri se tako, da se naredi seštevek točk (ang. story points) vseh končanih zahtev v prejšnji iteraciji. V prvi iteraciji se hitrost razvoja oceni približno glede na izkušnje razvijalca in na število dni, ki bo sodeloval

pri razvoju. Sledenje hitrosti razvoja skozi iteracije nam da približek za najbolj realno oceno dela, ki ga je sposobna razvojna skupina opraviti v iteraciji.

## 2.5 Zaključena zahteva

Zaključena zahteva je ključnega pomena pri visoko funkcionalni razvojni skupini. Koncept zaključene zahteve teži k temu, da je vsaka uporabniška zgodba, ki je razvita v iteraciji, v celoti dokončana in je pripravljena za implementacijo na produkcijsko okolje. Iteracija je uspešna takrat, ko gredo vse razvite zahteve skozi določene teste, ki jih definira razvojna skupina na začetku razvoja. S tem se prepreči prenašanje slabo razvitih zahtev v produkcijsko okolje in izboljša funkcionalnost razvojne skupine.

Nekaj primerov pravil iz prakse:

- testi enot,
- integracijski testi,
- napisana mora biti dokumentacija, ki razlaga delovanje implementirane zahteve,
- komentirana koda,
- isti izgled spletne strani v vseh brskalnikih,
- zahteva je bila testirana s strani drugega razvijalca in skrbnika metodologije.

## 2.6 Iteracija in njen potek

V tem podpoglavju opišemo iteracijo, njen potek in vse njene sestavne dele.

### 2.6.1 Iteracija

Iteracija (ang. Sprint) je določeno časovno obdobje, v katerem morajo biti določene funkcionalnosti produkta implementirane. Dolžina iteracije je v praksi najpogosteje nekje med enim tednom pa vse do enega meseca. Rezultat iteracije je delujoč del produkta, pripravljen za produkcijsko okolje. Iteracija je uspešna, ko so vse načrtovane zahteve za to iteracijo implementirane in testirane po konceptu zaključene zahteve.

### 2.6.2 Načrtovanje iteracije

Ko je začetni seznam zahtev definiran in so vse uporabniške zgodbe prioritizirane, se lahko načrtovanje iteracije (ang. Sprint planing) začne. Načrtovanje je ponavljajoči se proces, ki se izvaja, na sestanku načrtovanja iteracije (ang. Sprint planning meeting), ki poteka na začetku vsake iteracije.

Načrtovanje poteka v dveh delih, prvi del je ocenjevanje zahtev, drugi razdelitev dela med člani razvojne skupine. Iteracija se načrtuje na podlagi planirane hitrosti, ki jo je razvojna skupina sposobna opraviti v posamezni iteraciji. Torej za vsakega člana se oceni, koliko dela je sposoben opraviti v iteraciji, sešteje se vsota vseh članov in se dobi limit pri načrtovanju.

V prvem delu načrtovanja skrbnik izdelka predstavi cilje iteracije in zahteve z najvišjo prioriteto, ki se bodo implementirale v tej iteraciji. Člani razvojne skupine morajo razčistiti vse nejasnosti glede zahtev, saj jih morajo dobro razumeti, da bo ocenjevanje le-teh, kar se da realno. Sledi ocenjevanje predstavljenih zahtev, ki poteka za vsako zahtevo posebej. Pred ocenjevanjem zahteve se člani pogovorijo glede nje, izpostavijo morebitne težave in njihove rešitve. Ocena posamezne zahteve se lahko poda preko različnih tehnik ocenjevanja, med popularnejšimi je igra "poker planiranja"(ang. Planing poker).

V drugem delu načrtovanja se razvijalcem razdelijo zahteve iz seznama zahtev trenutne iteracije. Naloge se delijo glede na sposobnost posameznega razvijalca, torej če je neka zahteva zahtevnejša, in se ve da lahko pride do

kakšnega problema, se takšna naloga dodeli izkušenejšemu razvijalcu, na drugi strani se enostavnejše zahteve dodelijo mladim, manj izkušenim razvijalcem, da s tem dobijo potrebne izkušnje.

### 2.6.3 Poker planiranje

Poker planiranje je hitra, enostavna in učinkovita tehnika pridobivanja realne ocene za implementacijo zahteve. Za ocenjevanje se uporabljajo karte z določenimi števili, ki nam povedo uporabniške točke (ang. Story points), običajno so to števila 1, 2, 3, 5, 8, 13, 20, 40 in 100. Iz seznama zahtev se začnejo ocenjevati posamezne zahteve, cilj je doseči število delovnih enot, ki so se določile na začetku načrtovanja iteracije.

Iz seznama zahtev začnemo ocenjevanje posamezne zahteve, vsaka zahteva se še enkrat na kratko opiše in razčisti kakršnekoli nejasnosti, nato vsak član izbere karto s številko, ki se mu zdi najbolj približna številu delovnih enot, potrebnih za njeno implementacijo. Če se slučajno zgodi, da dve ali več kart izstopajo iz povprečja, lastniki teh kart podajo pojasnilo, zakaj menijo, da je za implementacijo potrebnega toliko več ali manj časa. Nato se ocenjevanje ponovi, postopek se ponavlja, dokler člani ne dosežejo soglasja oziroma dokler se ocene ne poenotijo. Glede na to, da vsak poda svojo oceno, brez vpliva drugih nanjo, je to zelo dober način za pridobitev realne ocene.

### 2.6.4 Dnevni sestanki

Dnevni sestanki se izvajajo vsak dan ob isti uri. Udeležijo se ga vsi odgovorni člani razvojne skupine. Običajno traja nekje 15 minut, načelo je "kratko in jedrnato". Dnevni sestanki niso namenjeni reševanju problemov, ampak poročanju. Vsak član razvojne skuupine poroča tri stvari:

- Kaj si naredil od prejšnega sestanka?
- Kaj boš delal do naslednjega sestanka?
- Kakšne težave imaš pri razvoju?

### 2.6.5 Pregled končane iteracije

Po vsaki končani iteraciji (ang. Sprint), se izvedeta dva sestanka. Prvi je pregled iteracije (ang. Sprint Review Meeting). Na tem sestanku razvojna skupina predstavi rezultate iteracije, torej vse zahteve, ki so bile implementirane in so pripravljene za produkcijo. Sestanka se poleg razvojne skupine in skrbnika metodologije, udeležijo tudi skrbnik izdelka in vsi ostali zainteresirani za projekt. Po končani predstavitvi, skrbnik izdelka poda svoje mnenje o rezultatu iteracije.

### 2.6.6 Ocene iteracije

Drug sestanek in zadnje dejanje v iteraciji je ocena iteracije (ang. Sprint Retrospective meeting). Na tem sestanku se oceni celotno delo skozi iteracijo. Pregledajo se dobre in slabe stvari, ki so se dogajale v iteraciji. Pogovori se o stvareh, ki so se delale oziroma niso delale v trenutni iteraciji in jih je potrebno ukiniti, začeti delati oziroma nadaljevati v naslednji iteraciji. Ker je vedno prostor za napredek, je razlog tega sestanka iskanje možnosti za izboljšanje razvoja in kakovost produkta.

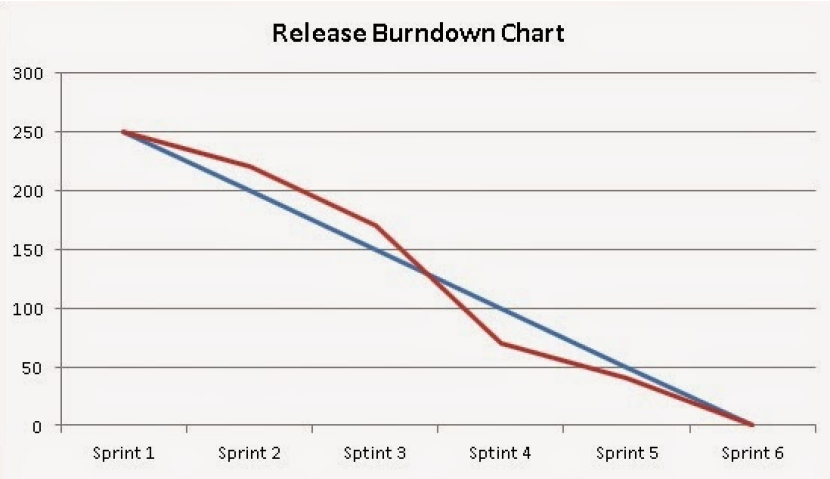
## 2.7 Spremljanje napredka na projektu

Za hitro prepoznavo preostalega dela, bodisi v trenutni iteraciji oziroma trenutni izdaji, uporabimo naslednja dva diagrama.

### 2.7.1 Diagram preostalega dela v trenutni izdaji

Diagram preostalega dela v trenutni izdaji (ang. Release burn-down chart) poda razvojni skupini vpogled in spremljanje napredka razvoja proti izdaji. Diagram se posodablja po vsaki končani iteraciji. Vertikalna os prikazuje število ur, horizontalna število iteracij. Glavna funckionalnost diagram je ta, da je hitro razvidno stanje razvoja projekta, ali bo izdaja zaključena do

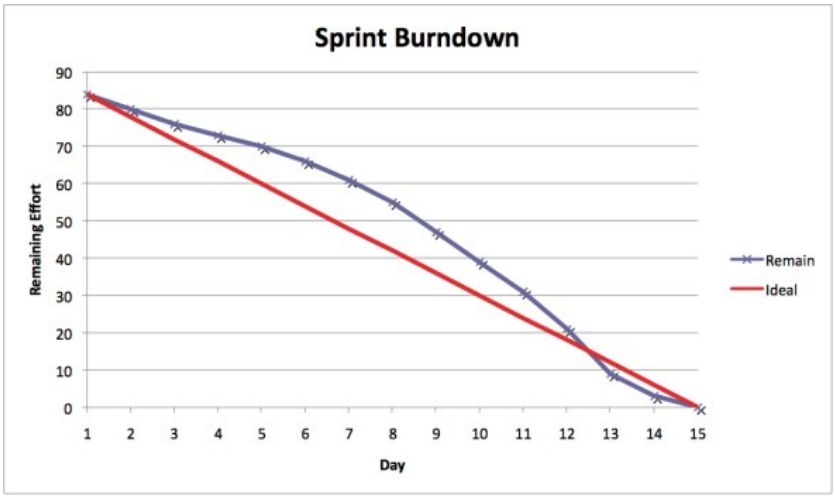
načrtovanega datuma. Slika 2.2 prikazuje diagram preostalega dela v iteraciji.




Slika 2.2: Prikazuje diagram preostalega dela v izdaji [16].

### 2.7.2 Diagram preostalega dela v trenutni iteraciji

Diagram preostalega dela v trenutni iteraciji (ang. Sprint burn-down chart) prikazuje preostalo število delovnih enot v tekoči iteraciji. Diagram se posodablja vsakodnevno in prikazuje napredek razvoja v trenutni iteraciji. Vertikalna os vsebuje število ur, horizontalna število dni v iteraciji. Namenjen je hitremu pregledu stanja iteracije, torej ali poteka vse tako kot je bilo načrtovano na začetku iteracije. Slika 2.3 prikazuje diagram preostalega dela v iteraciji.




Slika 2.3: Prikazuje daigram preostalega dela v iteraciji [15].

# Poglavje 3

# Uporabljene tehnologije

## 3.1 HTML

HTML (ang. Hyper Text Markup Language) je označevalni jezik, ki opisuje strukturno predlogo internetnih strani in aplikacij. Glavni gradniki teh so HTML elementi, ki jih predstavimo s tako imenovanimi značkami. Vsak element je sestavljen iz treh delov: iz začetne značke, opisa in končne značke. Začetna in končna značka sta obdani z znakoma <in >, med katerima je zapisano ime elementa (div, p, h, img, table). Znotraj začetne in končne značke je opis elementa[6, 5].

## 3.2 CSS

CSS je slogovni jezik, ki skrbi za prezentacijo spletnih strani. Večinoma se uporablja za nastavitev vizualnih pravil HTML elementov, ki se prikažejo na strani. Določa se lahko veliko različnih pravil, kot npr. bravo, velikost, obrobe, odmike, poravnave. Bistvo uporabe CSS je konsistenten način podajanja informacij o stilu spletnim dokumentom[2].

```css
body {
    background-color: lightblue;
}

h1 {
    color: white;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 20px;
}
```

Slika 3.1: Primer CSS kode.

## 3.3 JavaScript

JavaScript je visoko nivojski objektni skriptni programski jezik, ki se uporablja za grajenje dinamičnih spletnih strani in aplikacij. Sodi v skupino interpretiranih jezikov, pri katerih se programska koda ne prevede, ampak se po vrsti razčlenjuje, razčlenjeni ukazi se nato izvršijo.

Poleg HTML in CSS je jedro vsebinske proizvodnje svetovnega spleta in podprta pri vseh modernih spletnih brskalnikih. Uporablja se za komunikacijo z vsebino HTML dokumenta in za implementacijo interaktivnosti v spletne strani oziroma aplikacije, kot so razne animacije, dogodki, itd.

Tu je nekaj primerov za kaj se uporablja JavaScript pri spletnih vsebinah:

- prenos dokumentov/slik iz spleta
- nezaželjeno zapiranje brskalniškega okna
- delo z piškotki
- interakcija s strežnikom
- validacija vnešenih podatkov

Lahko se uporabi neposredno v HTML dokumentih ali ločenih ".js" datotekah, ki se nato neposredno povežejo s HTML dokumentom. V osnovi se je JavaScript izvajal samo na odjemalčevi strani, vendar se je v zadnjih letih s prihodom Node.js preselil tudi na zaledne sisteme. JavaScript ni uporabljen le pri spletnih tehnologijah, ampak je v uporabi tudi v PDF dokumentih in celo pri operacijskih sistemih.

Ker je popularen programski jezik, je zanj razvitih veliko uporabnih knjižnic, kar pripomore k lažjemu razvoju spletnih vsebin. Mi smo uporabili eno izmed popularnejših JavaScript ogrodij za razvoj dinamičnih spletnih aplikacij, AngularJS [7].

## 3.4 Programski paket MEAN

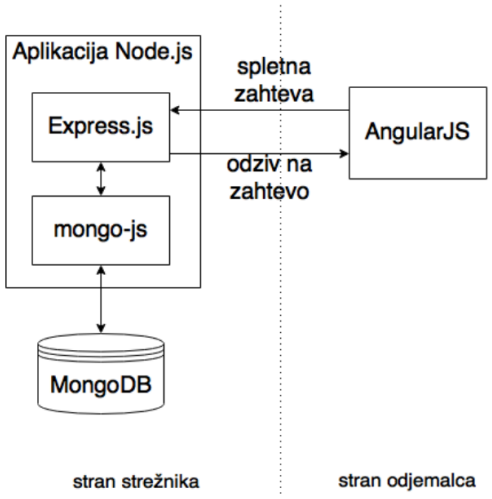
Kot glavni gradnik sistema aplikacije smo uporabili programski paket MEAN, ki ga sestavljajo podatkovna baza MongoDB, Express.js, AngularJS in NodeJS tehnologije. Za ta programski paket smo se odločili zaradi vseh pozitivnih stvari, ki jih vsaka posamezna tehnologija prinaša na svojem področju. Doprinosi posameznih tehnologij so navedeni pri podrobnejših opisih tehnologij. Vsebina podpoglavja je povzeta iz vsebin [18, 17]. Slika 3.2 prikazuje strukturo programskega paketa MEAN.

### 3.4.1 MongoDB

Za hranjenje podatkov smo se odločili za uporabo podatkovne baze MongoDB, saj ponuja veliko lastnosti, ki so bile pri grajenju aplikacije uporabne. MongoDB je odprtokodna NoSQL baza podatkov, ki je bila razvita s strani podjetja MongoDB Inc. in objavljena pod kombinacijo licenc Apache in GNU Affero General Public.

To, da je MongoDB NoSQL, pomeni, da se izogiba klasični podatkovno relacijski strukturi in podatke hrani v BSON (ang. binary Javascript object notation) obliki. Slika 3.3 prikazuje primer objekta MongoDB.

MongoDB poleg dokumentno orientirane strukture podpira tudi dinamične




Slika 3.2: Struktura programskega paketa MEAN

sheme, torej omogoča lažje dodajanje novih atributov v bazo, kar omogoča enostavnejšo in hitrejšo integracijo v določene dele aplikacije.

Ravno tako kot pri SQL bazah MongoDB podpira, tako primarno kot sekundarno indeksiranje, kar pripomore k hitrejšemu delovanju celotne baze. Ena izmed prednosti podatkovne baze MongoDB je ta, da podpira "ad-hoc" poizvedbe, kar pomeni, da lahko poizvedbe išče po vseh poljih, lahko so omejene za določeno območje oziroma so lahko tudi poizvedbe z regularnimi izrazi. Poleg iskalnih lastnosti lahko poizvedbe vračajo tudi samo določena polja v dokumentu ali vsebujejo uporabniško definirane JavaScript funkcije.

Zelo koristna lastnost podatkovne baze MongoDB je ta, da vsakemu objektu doda 12-bitni heksadecimalni niz znakov, ki enolično določa objekt po celotni bazi, kar smo koristili tudi na čelnem sistemu aplikacije. Ker smo podatke prenašali preko API-jev v JSON obliki, nam je ta oblika podatkov v bazi prišla zelo prav, saj pred shranjevanjem oziroma branjem ni bilo veliko dela s preoblikovanjem le-teh [8, 9].

```
{
  "data": {
    "versions": [
      {
        "data": {
          "viewPravice": [],
          "editPravice": [],
          "znacke": [],
          "povezave": [],
          "ponavljajoci": [],
          "placljiv": "Ne",
          "disabled": "Ne",
          "name": "Diplomsko delo",
          "pid": "12321421",
          "poslovnaEnota": "Projekt",
          "odgovornaOseba": "Janez Novak",
          "tip": "Interni projekt",
          "program": "Razvoj",
          "planUr": {
            "2017": "360"
          }
        }
      }
    ]
  },
  "internaSifra": "12321421",
  "id": "5977ae8cba0d5824ff71f6e5"
}
```


Slika 3.3: MongoDB objekt

### 3.4.2 Node.js

Zaradi enostavne uporabe in želje po hitrem, optimiziranem sistemu smo se za programsko okolje na zalednem sistemu odločili za Node.js. Kot osnovo Node.js uporablja programski jezik JavaScript.

Node.js implementira dogodkovni model in vhodno-izhodni sistem, kateri ob morebitnem čakanju na izvršitev dogodka ne zaklene aplikacije, ampak omgoči da se dogodki izvajajo paralelno. Za zaznavanje uspešnega oziroma neuspešnega dogodka uporablja signale povratnega klica, s pomočjo katerih posrkbi za dobro pretočnost in razširljivost samega sistema. To ga naredi

popolnega za izmenjavo velike količine podatkov v realnem času med več napravami, ki uporabljajo različne operacijske sisteme.

Okolje je zgrajeno, tako da omogoča uporabo večjega števila različnih knjižnic, ki jih Node.js podpira. Uporaba teh, ponastavi in izboljša delovanje celotne aplikacije, saj so večinoma narejene tako, da so kar se da optimizirane za uporabo. Za dodajanje, posodabljanje oziroma brisanje knjižnic iz sistema skrbi NPM upravljalec knjižnic. NPM uporablja globalno datoteko imenovano package.json, v kateri so definirane vse uporabljene knjižnice, s tem se izognemo prenašanju prostorsko velikih knjižnic pri distribuciji končne aplikacije. Ko postavljamo končno aplikacijo na druge sisteme, z ukazom npm install prenesemo vse knjižnice, definirane v package.json iz spletnega repozitorija[10, 11]. Slika 3.4 prikazuje primer datoteke *package.json*.

```
{
  "name": "app",
  "version": "0.0.0",
  "description": "Generated app app.",
  "main": "server/index.js",
  "private": true,
  "dependencies": {
    "angular-material-data-table": "^0.9.14",
    "body-parser": "~1.5.0",
    "composable-middleware": "^0.3.0",
    "compression": "~1.0.1",
    "connect-mongo": "^0.4.1",
    "ejs": "~0.8.4",
    "errorhandler": "~1.0.0",
    "express": "~4.9.0",
    "gulp-jsdoc": "^0.1.5",
  },
```

Slika 3.4: Primer vsebine package.json dokumenta

### 3.4.3 Express.js

Express.js je odprtokodno minimalistično in fleksibilno ogrodje za Node.js, ki ponuja množico robustnih funkcij za grajenje spletnih in mobilnih aplikacij.

Na zalednem sistemu pomaga pri organizaciji modela MVC (ang. Model View Controller). Skrbi, da nam ni potrebno določenih metod programirati vse od začetka, ampak jih lahko samo vključimo v aplikacijo, primeri teh metod so usmerjanje, nadziranje zahtev, postavljanje okolja na strežnik, s tem nam prihrani veliko časa, ki ga lahko posvetimo grajenju drugih delov aplikacije[3].

### 3.4.4 AngularJS

AngularJS je odprtokodno JavaScript ogrodje za razvoj dinamičnih spletnih aplikacij. Preverjene prakse na strani strežnika vpeljuje na stran odjemalca in s tem zmanjšuje obremenjenost zalednega sistema. S tem ko je dodan nov nivo abstrakcije med razvijalcem in pogostimi nalogami, ki jih mora razvijalec implementirati v spletni aplikaciji, nam olajša njeno izdelavo.

Angular implementira MVC (angl. Model View Controller) model, torej je logika sistema razdeljena na tri dele, prvi je model, ki predstavlja podatke, ki se bodo prikazovali, drugi je pogled, ki poskrbi za prikaz podatkov in vsebine, tretji del je kontroler, ki skrbi za obdelavo podatkov. Večinoma se AngularJS uporablja za grajenje SPA (angl. Single Page Application) aplikacij, to so aplikacije, katerih vsebina je vidna znotraj ene spletne strani.

Glavna značilnost AngularJS je ta, da podpira dvosmerno vezavo podatkov (ang. Two-way data bining), kar pomeni, da poskrbi za samodejno sinhronizacijo podatkov med modeli in pogledi.

AngularJS deluje, tako da iz HTML vsebine, zazna Angular atribute z oznako “ng-”, ki mu povedo, kako naj poveže vhodno - izhodne dele strani na model, ki ga predstavljajo spremenljivke.

Nekaj pomembnejših direktiv, ki jih uporablja AngularJS:

- ng-app: definira predlogo kot del AngularJS spletne aplikacije,
- ng-controller: določuje, kateri kontrolnik nadzoruje HTML element,
- ng-repeater: iteracija HTML elementa čez podan seznam podatkov[1].

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>

<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
    $scope.firstName = "Janez";
    $scope.lastName = "Novak";
});
</script>

</body>
</html>
```

Slika 3.5: Primer kode AngularJS in HTML

## 3.5 Git

Git je sistem za upravljanje z izvorno kodo. Je eden najbolj razširjenih sorodnih sistemov na področju razvoja programske opreme. Glavna naloga gita je spremljanje sprememb kode v določenih datotekah. Najbolj je uporabljen pri programskem razvoju, za verzioniranja programske kode. To pomeni, da skrbi za brisanje, dodajanje in združevanje kode v datotekah, na katerih so bile opravljene spremembe.

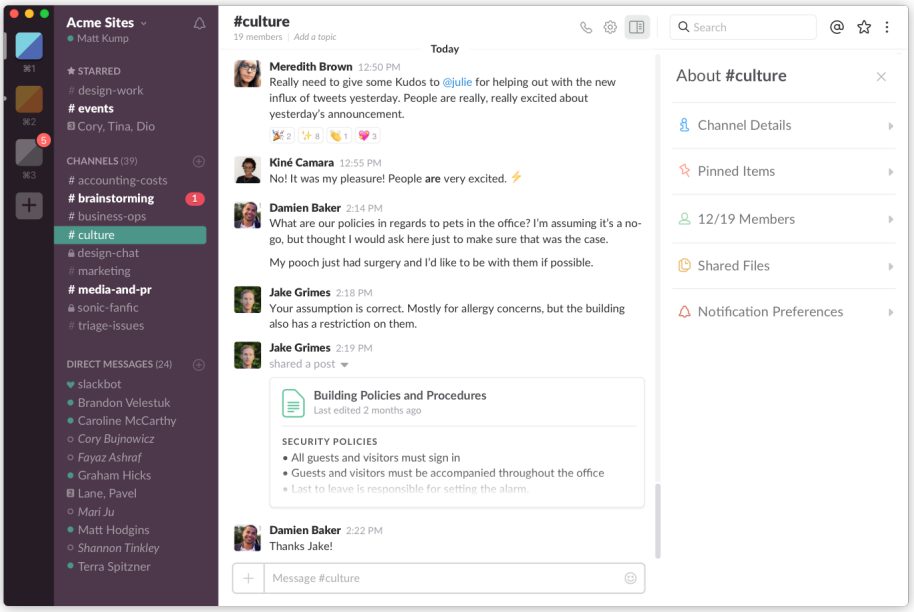
Datoteke se hranijo v spletnih repozitorijih, na katere se redno objavljajo verzije kode. Git je zelo primeren za uporabo pri razvoju projekta, na katerem dela več razvijalcev, saj poskrbi, da ko dva razvijalca objavljata spremembe na isto datoteko, ne pride do izgube kode. Git zazna in sporoči razvijalcu, da je prišlo do konflikta v kodi, ki ga je treba rešiti. Naš git repozitorij se je nahajal na internih strežnikih podjetja[4].

## 3.6 Slack

Za komunikacijo med razvojno skupino se je uporabljal Slack. Slack je ogrodje pripomočkov za sodelovanje znotraj razvojne skupine. Ponuja veliko koristnih funkcij, skupinske sobe oziroma kanali, ki so lahko fiksni za

določeno skupino v podjetju, organizirane na podlagi teme pogovora ali zasebne skupine.

Vsa vsebina (dokumenti, pogovori, člani) znotraj slacka je lahko iskana preko iskalnega okna, tako se lažje in hitrejše dostopa do želene vsebine. Gre za integracijo z ostalimi sistemi, saj lahko slack povežeš z drugimi sistemi in si tako obveščen, ko se določen dogodek zgodi v tem sistemu, npr. povezava z gitom, vsakič, ko se objavi nova verzija, te slack obvesti




Slika 3.6: Izgled Slack ogrodja [14].

# Poglavje 4

# Razvoj spletne aplikacije

## 4.1 Opis aplikacije

*Aplikacija Plan* je namenjena urejanju podatkov projektov, načrtovanju finančnih planov projektov in spremljanje realizacije le-teh. Uporabljena je s strani ožjega kolegija v podjetju. Namen je uporabniku prikazati projekte, pri katerih sodeluje, na kar se da enostaven način, tako da ima uporabnik čim boljši pregled nad njimi. Uporabnikom je omogočeno urejanje podatkov projekta. Pri vseh projektih so osnovni podatki isti. Poleg osnovnih podatkov lahko uporabnik ureja podatke, ki so odvisni od tipa projekta (Storitev, Produkt, Interni projekt, Finančni plan), te bomo podrobneje predstavili malo kasneje.

Prikaz in urejanje finančnih planov sta omejena na posamezno izbrano leto, za katero se prikaže prihodke in odhodke po vseh mesecih. Poleg prikaza finančnih planov prikažemo grafe realizacije le-teh. Torej prikažemo graf, ki nam pove, kakšna je bila realna uspešnost projekta glede na planirano uspešnost. Poleg projekta se nadzira tudi administracija le-tega. Pri vsakem projektu se lahko uprabniku doda pravice nad njim (vpogled, urejanje), malo podrobneje bomo o pravicah govorili v nadaljevanju. Zaradi nadzora nad podatki, se je v aplikacijo Plani dodalo kontrolo za spremljanje sprememb na projektu. O vsaki spremembi na projektu se shranjuje njene informacije.

Shrani se, kdo je naredil spremembo, kdaj in kakšna je bila le-ta.

### 4.1.1 Projekt

Za vsak projekt so definirani podatki, ki jih potrebujemo v tej aplikacije ali v sistemih, na katere je aplikacija povezana. Podatke projekta delimo v naslednje štiri kategorije: osnovni podatki, podrobni podatki, finančni plan ter administracija.

Osnovne podatke sestavljajo naslednji podatki: ime projekta, interna šifra, ki enolično določa projekt in njegov stroškovni nosilec, poslovna enota, del katere je projekt, odgovorna oseba nad projektom, ali je projekt plačljiv, program, podprogram, značke, tip projekta (storitev, produkt, interni projekt, finančni plan).

Podrobni podatki so dinamično prikazani, saj so odvisni od tipa projekta. Projekt tip storitev je projekt, ki se ga razvija po naročilu. Tipično ima jasno definiran začetek in konec. Projekt tipa produkt, je projekt, ki se ga začne razvijat, brez da bi se vedelo ali ima projekt končne kupce. Med samim razvojem se že išče naročnike in en projekt lahko prodaš različnim kupcem. Interni projekt se izvaja znotraj podjetja, tipično nima prihodkov. Namenjen je dodajanju vrednosti v podjetju. Projekt tipa finančni plan, je projekt, ki se uporablja za beleženje stroškov podjetja (plače, sistemski stroški,...). Finančni plani vsebujejo podatke o planiranih prihodkih in odhodkih za vsak mesec v določenem letu. Administracija vsebuje podatke o tem, kateri uporabniki imajo pravice nad projektom.

## 4.2 Postavitev okolja

Razvoj aplikacije smo začeli s postavitvijo okolja. Namestili smo vsa potrebna ogrodja iz programskega paketa MEAN, postavili smo podatkovno bazo MongoDB, v projekt dodali knjižnice, ki jih potrebujemo, ter postavili in med seboj povezali zaledni in čelni sistem. Po nameščenem programskem okolju smo začeli z delom.

### 4.2.1 Podatkovni model in verzioniranje objekta

Zaradi nejasnega začetnega modela in dinamičnega dodajanja atributov smo se zaradi podpore želenih funkcij odločili za podatkovno bazo MongoDB. Naš začetni podatkovni model projekta je vseboval dva atributa, in sicer data in interno šifro projekta. Data atribut je objekt, ki vsebuje vse podatke o projektu in njegovih verzijah. Interna šifra je atribut tipa niz in je namenjen za preverjanje enoličnosti atributa skozi aplikacijo in skozi vse sisteme v podjetju, s katerimi so finančni podatki projekta povezani. Poleg teh dveh definiranih atributov, podatkovna baza MongoDB avtomatsko doda 12-bitni heksadecimalni niz znakov kot ID objekta, katerega uporabljamo za enolično (nikoli se ne spremeni) določanje projekta skozi aplikacijo.

Velikokrat se v aplikacijah kjer imajo različni uporabniki dostop do skupnih podatkov zgodi, da se določeni podatki spremenijo in se ne ve kdo in kdaj je te podatke spremenil. Pri aplikaciji, kot je naša, zna biti to velik problem, saj so podatki ključnega pomena za aplikacijo in druge sisteme, na katere je aplikacija povezana. Da bi se izognili fizičnemu povpraševanju sodelavcev po podjetju, smo se odločili za implementacijo verzioniranja podatkov. Vse verzije projekta smo hranili v seznamu, to pomeni, da smo ob vsaki posodobitvi podatkov o projektu, te shranili kot nov podobjekt v seznamu verzij projekta. Zadnjo verzijo projekta smo dobili tako, da smo vzeli zadnji objekt iz seznama verzij projekta.

Takšen način verzioniranja pride prav pri prikazu zgodovine sprememb, saj imaš na enostaven in hiter način dostop do vseh prejšnjih verzij. Ker aplikacija ne bo obdelovala velike količine podatkov, s takšnim načinom verzioniranja ne bo prihajalo do težav z velikostjo posameznih objektov.

### 4.2.2 Priprava REST API

Za pridobivanje, dodajanje, posodabljanje in brisanje podatkov smo uporabili spletno storitev REST (angl. Representational State Transfer) API (angl. Application Programming Interface). REST API deluje na način, da odje-

malec pošlje zahtevo na strežnik, ta mu odgovori s številko statusa oziroma v nekaterih primerih tudi s podatki. Zahteve se pošiljajo prek HTTP oziroma HTTPS protokola. Podatke smo pošiljali in prejemali v JSON (JavaScript Object Notation) formatu. V naši aplikaciji so uporabljene naslednje REST API metode:

- z metodo GET dobimo vse objekte iz strežnika, z dodanim parametrom ID, pa le objekt z danim IDjem,
- z metodo POST podatke pošljemo na strežnik. Podatke pošljemo na način, da jih dodamo v telo (ang. body) zahteve,
- z metodo PUT posodobimo podatke objekta s podatki, ki jih pošljemo v zahtevi,
- z metodo DELETE izbrišemo objekt, katerega ID je dodan kot parameter.

### 4.2.3 Prenos podatkov

Naredili smo skripto, ki je prenesla vse podatke iz podatkovne baze drugega sistema v podatkovno bazo aplikacije. V skripti se povežemo na podatkovno bazo zunanjega sistema in poženemo poizvedbo, ki vrne vse projekte. Vse podatke, dobljene iz poizvedbe, smo morali pred shranjevanjem v podatkovno bazo malo prilagoditi strukturnim potrebam.

## 4.3 Implementacija zahtev

Implementacija zahtev je bila ločena na tri glavne poglede: začetni pogled oziroma prikaz projektov, pogled urejanje projekta in pogled urejanje večih projektov. Preden začnemo opisovati implemnatcijo zahtev, bomo predstavili administracijo aplikacije in administracijo nad posameznim projektom.

### 4.3.1 Administracija aplikacije

Ker gre za interno aplikacijo, je njena uporaba omejena na privatno omrežje podjetja. Za prijavo se uporablja domenski uporabniški račun, ki se uporablja za sisteme znotraj podjetja. Za nadzor nad domenskimi uporabniškimi računi podjetje uporablja LDAP programski protokol, ki se uporsblja za poizvedovanje in spreminjanje imeniških storitev, ki tečejo preko TCP/IP protokola. Na podlagi domenskega računa uporabniku dodeljujemo pravice nad aplikacijo. Na zalednem sistemu smo vzpostavili povezavo na LDAP strežnik, na katerega pošiljamo poizvedbe o uporabnikih. Pri vsaki prijavi smo naredili poizvedbo na LDAP strežnik, z zahtevo o tem, ali je uporabnik res del domenskega imenika in če je del, ali ima kakšne pravice nad aplikacijo.

Uporabnik lahko ima naslednje tri vrste pravic na aplikacijo: globalni administrator, član ožjega kolegija ter navaden uporabnik. Globalni administrator in član ožjega kolegija imata pravice za vpogled nad vsemi projekti, globalni administrator lahko v osnovi pri vsakem projektu ureja le ime projekta, interno šifro projekta, status projekta (zaključen/aktiven) in dodaja pravice nad projektom sebi oziroma ostalim uporabnikom. Navaden uporabnik ima dostop v aplikacijo, vendar nima nobenih globalnih pravic, pravice ima lahko samo nad posameznimi projekti.

Vsakemu uporabniku aplikacije se je v podatkovni bazi aplikacije dodal objekt z domenskim spletnim naslovom in mu je bil avtomatsko dodeljen ID (lastnost podatkovne baze MongoDB). Po vsaki uspešni poizvedbi na LDAP strežnik, se je na podlagi spletnega naslova naredila poizvedba v podatkovno bazo, z dobljenim objektom smo naredili poizvedbo, če ima ta uporabnik že dodeljen dostopni žeton, če ga nima, smo ga ustvarili in shranili v bazo. Življenjska doba dostopnega žetona je omejena na dva tedna, torej po dveh tednih dostopni žeton ni več veljaven in se uporabniku dodeli novega. Na takšen način smo se rešili pogostega povezovanja na LDAP strežnik. S tem smo optimizirali varnostno preverjanje uporabnika.

### 4.3.2 Pravice uporabnika nad projektom

Nad projektom lahko ima uporabnik poleg globalnih pravic, dve vrsti pravic, pravico za vpogled nad projektom, ki mu omogoča pogled nad vsemi podatki v projektu, vendar jih ne more urejati. Lahko ima administratorske pravice nad projektom, ki mu omogočajo urejanje vseh podatkov, razen imena projekta, interno šifro in status projekta (aktiven/zaključen), kot smo omenili lahko te tri podatke urejajo le globalni administratorji aplikacije.

### 4.3.3 Začetni pogled oz. prikaz projektov

Implementacijo zahtev smo začeli z oblikovanjem glavne strani, ki prikazuje podatkovno tabelo z vsemi projekti v podjetju. Projekte smo v tabeli prikazali s pomočjo AngularJS atributa ng-repeat, ki omogoča iteracijo skozi seznam objektov in izpis atributov posameznega objekta.

V tabeli je omogočeno urejanje po stolpcih, in sicer po imenu projekta, poslovni enoti in odgovorni osebi. Pri vsakem projektu v tabeli je v zadnjem stolpcu dodana ikona "urejanje", ki uporabnika preusmeri na stran za urejanje projekta.

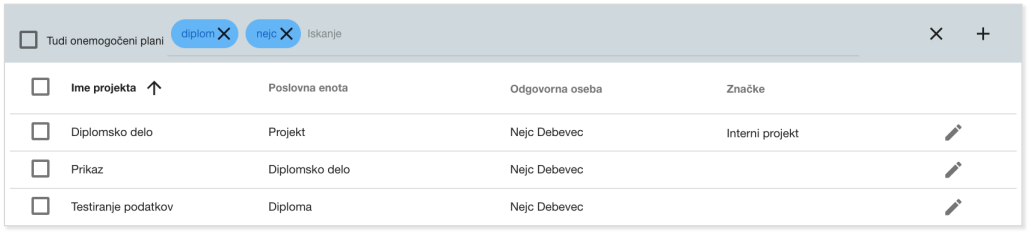
Nad tabelo je postavljen iskalnik med projekti. Ta deluje na takšen način, da na podlagi gnezdenih iskalnih nizov, išče po vseh stolpcih tabele. Gnezdenje je mišljeno tako, da se za vsak iskalni niz naredi filtracija na rezultatih iskanja predhodnih nizov. Vse trenunto uporabljene iskalne nize hranimo v seznamu filtrov. Vsak dodan iskalni niz v seznam filtrov, bo dodan tudi v URL naslov, kar uporabniku omogoča deljenje iskalnih rezultatov z drugimi uporabniki, ali pa pripravo vnaprej določenih filtrov. Isti princip delovanja je pri brisanju iskalnih nizov is seznama, iskalni niz se bo iz URL-ja izbrisal in rezultat iskanja bo prilagojen na iskalne nize, ki so ostali v seznamu filtrov.

Primer uporabe: na sestanku ožjega kolegija v podjetju, želijo pregledati vse projekte z vsebovanim nizom EU (evropski projekt), iz poslovne enote "Projekti", katerih odgovorna oseba je "Janez Novak", tako si lahko tehnični direktor vnaprej pripravi filtre in ne moti sestanka z vmesnim iskanjem pro-

jektov.

AngularJS omogoča implementacijo svojih filtrov na že dodan atribut ng-repeat,zato smo pripravili svoj filter. Filtri na atributih ng-repeat, delujejo na način, da se v krmilnik filtra pošlje celoten seznam objektov, tam se izvede celotna logika filtra in nato vrne prefiltrirane podatke. Ob vsaki spremembi podatkov oziroma iskalnih nizov se zgodi filtracija seznama objektov.

V glavi tabele so poleg iskalnika še tri ikone, prva je namenjena brisanju filtrov iz iskalnika, druga je namenjena dodajanju novih projektov in tretja, ki se dinamično prikazuje, je namenjena urejanju finančnih planov več projektov hkrati. Na začetku glave tabele je potrditveno polje (ang. check-box), ki prikaže tudi vse že zaključene projekte. Tem smo spremenili stil, barvo pisave smo spremenili na oranžno-rumeno, da se lahko ločijo od aktivnih projektov. To funkcionalnost potrditvenega polja smo isto kot iskalnik implementirali kot filter na celoten seznam. V krmilniku filtra smo preverjali, ali je projekt aktiven in če ni bil, ga nismo vrnili v seznamu. Slika 4.1 prikazuje tabelo s projekti.




Slika 4.1: Izgled tabele s projekti.

Istočasno, ko smo oblikovali stran, je bilo potrebno pripraviti programski vmesnik za REST metodo GET, ki bo vračala seznam vseh projektov. Na programskem vmesniku smo pri poslani zahtevi preverjali njeno veljavnost, to smo storili tako, da smo preverili, če je dostopni žeton res veljaven. Če žeton ni bil v lasti uporabnika, ki je poslal zahtevo, smo jo zavrnili. Prav tako smo na podlagi uporabnikovih pravic, vračali seznam projektov. Če je bil uporabnik globalni administrator ali član ožjega kolegija, je do-

bil seznam vseh projektov, če uporabnik ni imel posebnih pravic, je dobil seznam projektov, nad katerimi je imel pravice za vpogled oziroma za urejanje. Poskrbeti smo morali, da smo seznam projektov napolnili le z zadnjimi verzijami projektov, nad katerimi ima uporabnik pravice, torej, za vsak projekt smo iz seznama verzij projekta, dostopali do zadnjega objekta v seznamu. Ko smo nastavili logiko za preverjanje pravic na projektih, je bila metoda GET pripravljena in dostopna na naslednjem URL-ju: https://plan-api.comland.si/Plans/allLastVersion.

### 4.3.4 Prijavno okno

Pri implementaciji prijavnega okna nismo kompliciral, oblikovali smo ga preprosto, držali smo se osnovnih, lahkih barv. Ob uspešni prijavi smo uporabnika preusmerili na glavno stran (prikaz projektov), ob neuspešni prijavi, smo uporabniku prikazali sporočilo, ki je vsebovalo, za kakšno napako gre, bodisi za napačne uporabniške podatke ali napako na strežniku. Slika 4.2 prikazuje izgled prijavnega okna.

### 4.3.5 Vpogled/urejanje projekta

Do pogleda urejanja projekta lahko dostopamo preko glavnega pogleda, kjer izberemo projekt, ki bi ga radi urejali. Slika 4.3 prikazuje izgled strani za urejanje projekta.

Kot je razvidno iz slike, so podatki razporejeni po sekcijah. Na podlagi pravic uporabnika smo polja zaklepali oziroma odklepali. Sekcija osnovnih podatkov vsebuje podatke o imenu projekta, interni šifri in statusu projekta (aktiven/zaključen), ki so na voljo za urejanje le globalnim administratorjem aplikacije. Ostali podatki v sekciji so za urejanje na voljo le uporabnikom, ki imajo administracijske pravice nad projektom. Za zaklepanje polj smo uporabili atribut ng-disabled, ki smo ga namestili na vsa polja, potem smo preko krmilnika pogleda, upravljali zaklepanje polji. To smo storili, tako da smo izkoristili AngularJS funkcijo, ki omogoča dvosmerno povezavo spremenljivk,

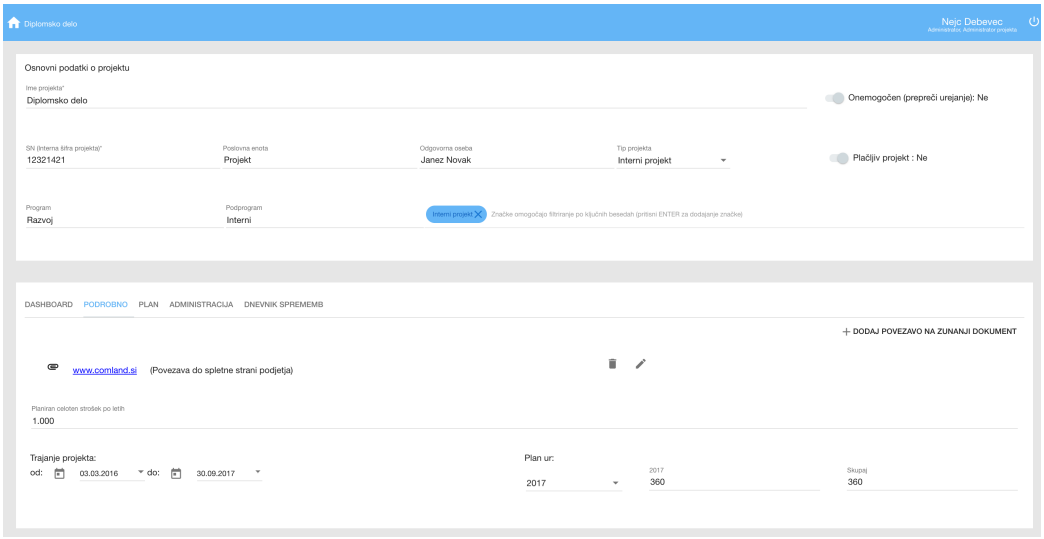

Slika 4.2: Izgled prijavnega okna.

torej iz pogleda na krmilnik in obratno. Tako samo določili globalne spremenljivke, ki so bile nastavljene glede na pravice uporabnika, te smo povezali z atributom ng-disabled.

*Zavihek dashboard* vsebuje grafe, ki prikazujejo različne analize finančnih podatkov. Samo implementacijo in prikaz grafov bomo podrobneje opisali v nadaljevanju.

Najpomembnejši zavihek je zavihek *Plan*, ki vsebuje planirane finančne podatke za projekt. Tu lahko uporabnik za izbrano leto dodaja oziroma spreminja prihodke in odhodke projekta po mesecih. Za urejanje prihodkov in odhodkov projekta smo uporabili tabelo. Na sliki 4.4 lahko vidite izgled tabele za planiranje prihodkov in odhodkov.

Tabelo smo pripravili kot direktivo (ang.directive), ki bo vsebovala HTML predlogo (tabelo) in njen krmilnik. AngularJS omogoča implementacijo direktiv, ki ponujajo grajenje svojih "HTML elementov", katere lahko uporabljamo skozi celotno aplikacijo. Direktivo smo uporabili tudi pri urejanju




Slika 4.3: Izgled strani za urejanje projekta

finančnega plana več projektov. S tem smo poskrbeli za konsistenčen prikaz podatkov skozi aplikacijo.

Samo implementacijo direktive smo začeli s postavitvijo HTML predloge. V to smo vključili tabelo, nad katero se nadzirajo vsa vnosna polja za spremembe, ob vsaki spremembi smo uporabniku omogočili, da se tabela ponastavi na začetne vrednosti, ki so bile pred urejanjem. To smo naredili s pomočjo uporabe AngularJS knjižnice angular-input-modified, ki omogoča nadzor vseh sprememb nad določenim objektom. Tako smo pred vsakim začetkom urejanja, podali dan objekt. Knjižnica ta objekt shrani, zato da se lahko ob morebitni spremembi vrnejo vrednosti podatkov v objektu na začetno stanje. Med urejanjem lahko začetno stanje nastavimo na trenutno stanje projekta z ukazom "$setPristine()" oziroma ponastavimo stanje na začetno z ukazom "$reset".

Implementirali smo dodajanje ponavljajočih se finančnih planov. To uporabniku omogoči, da finančne plane, ki so enaki skozi neko obdobje, doda preko enega pojavnega okna in mu ni potrebno za vsak mesec posebej vpisovati prihodkov in odhodkov. Da je uporabniku jasno, kateri so finančni plani

vneseni preko tabele in kateri preko funkcije ponavljajočih finančnih podatkov, smo ponavljajoče se finančne podatke prikazali v posamezni tabeli na dnu strani. Za prikaz seštevka vseh finančnih planov (iz tabele finančnih planov in tabele ponavljajočih finančnih planov) glede na prihodke in odhodke v mesecu smo dodali še eno tabelo. Ker AngularJS podpira dvosmerno povezavo podatkov se v tabeli seštevki prikazujejo ob spremembah v obeh tabelah. V krmilniku pogleda smo pripravili funkcije, ki opravljajo logiko seštevanja podatkov za prikaz v skupni tabeli. Nad vsemi polji v tabeli smo dodali denarni filter, ki podatke v poljih prikazuje v denarnem formatu. Slika 4.4 prikazuje glavni tabeli prikaza planiranih finančnih podatkov.

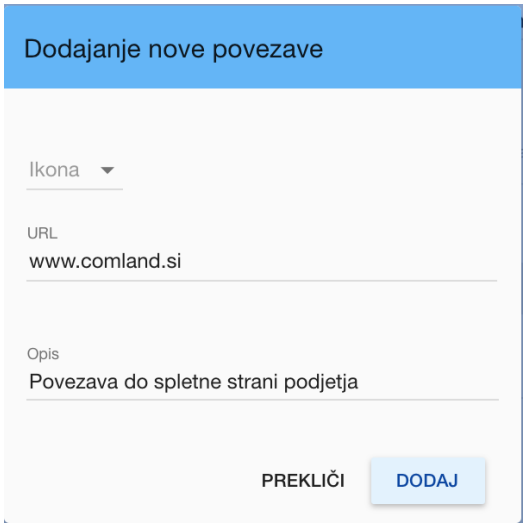
2017

| Mesec | Prihodki | Odhodki |
|---|---|---|
| **Januar** | 150 | -200 |
| **Februar** | 100 | -200 |
| **Marec** | 150 | -200 |
| **April** | 100 | -200 |
| **Maj** | 150 | -200 |
| **Junij** | 100 | -200 |
| **Julij** | 150 | -200 |
| **August** | 1.000 | -200 |
| **September** | 1.500 | -200 |
| **Oktober** | 100 | -300 |
| **November** | 150 | -300 |
| **December** | 100 | -300 |
| **Skupaj** | **3.750,00** | **-2.700,00** |

| Mesec | Prihodki | Odhodki |
|---|---|---|
| Januar | 150 | -100 |
| Februar | 100 | -100 |
| Marec | 150 | -100 |
| April | 100 | -100 |
| Maj | 150 | -100 |
| Junij | 100 | -100 |
| Julij | 150 | -100 |
| August | 1.000 | -100 |
| September | 1.500 | -100 |
| Oktober | 100 | -200 |
| November | 150 | -200 |
| December | 100 | -200 |
| **Skupaj** | **3.750,00** | **-1.500,00** |

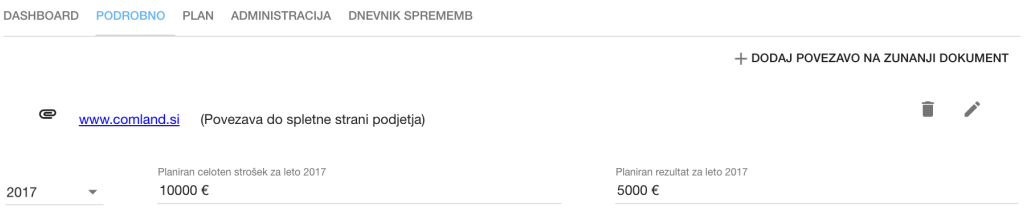
Slika 4.4: Levo tabela seštevkov planov in ponavljajočih planov, desno tabela planov.

*Zavihek Podrobno*, vsebuje podatke, ki se prikazujejo dinamično glede na tip projekta. Vsi tipi projekta imajo en skupen podatek in to je povezave na reference projekta, npr. spletna stran projekta, povezava na razpise. Za dodajanje referenc smo pripravili pojavno okno (ang. Pop-up), ki uporabniku nudi boljšo preglednost nad dodajanjem novih povezav. Slika 4.5 prikazuje izgled pojavnega okna.

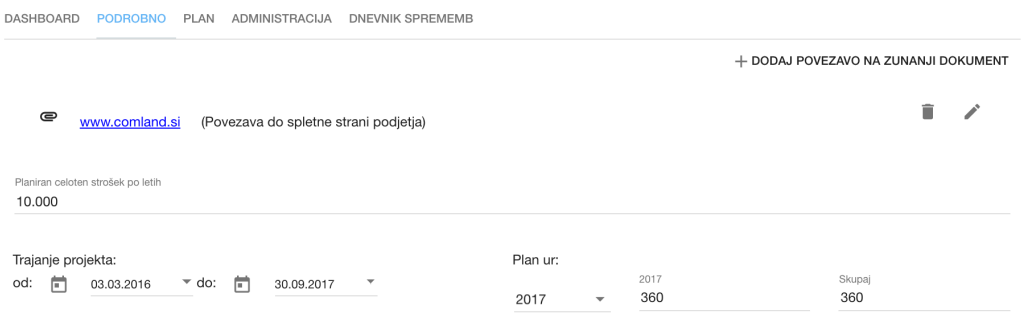



Slika 4.5: Izgled pojavnega okna za dodajanje nove povezave

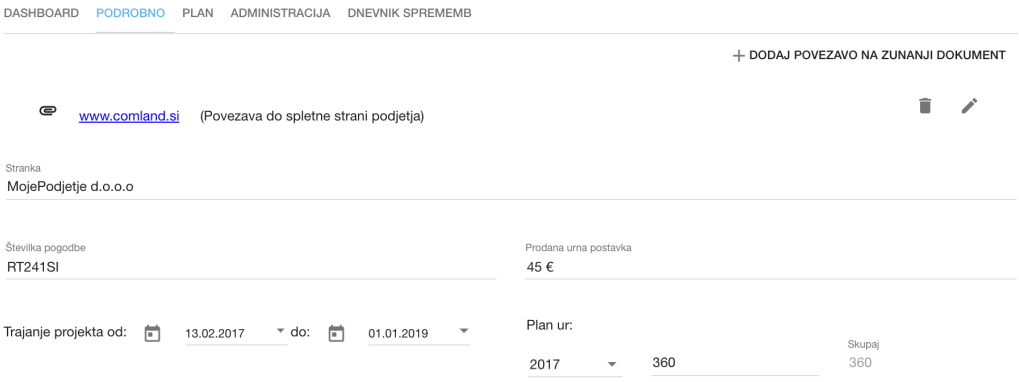
Izgled zavihka *Podrobno* glede na tip projekta prikazujejo slike: 4.6, 4.7, 4.8. Tip projekta Finančni plan vsebuje samo tabelo o referencah projekta, ki je prikazana že pri ostalih tipih projekta.




Slika 4.6: Izgled zavihka *Podrobno* glede na tip projekta Produkt

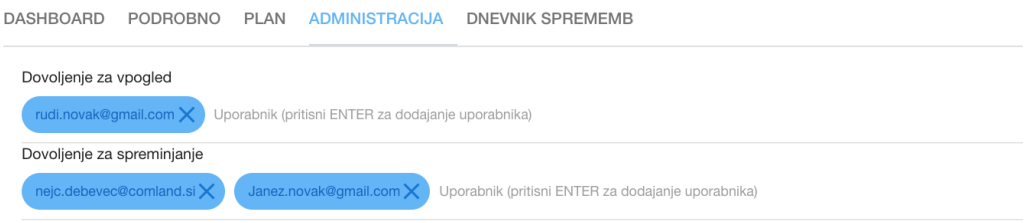



Slika 4.7: Izgled zavihka *Podrobno* glede na tip projekta Interni Projekt




Slika 4.8: Izgled zavihka *Podrobno* glede na tip projekta Storitev

*Zavihek Administracija* vsebuje dva seznama, ki sta ista kot seznam v iskalniku, torej prikazana z *md-chips* komponento, ki jo vpeljuje Angular Material. Ta na interaktiven način omogoča dodajanje in prikazovanje seznama uporabnikov. Prvi seznam prikazuje uporabnike, ki imajo pravico za pogled nad projektom, drugi seznam pa uporabnike z administratorskimi pravicami nad projektom. Izgled zavihka *Administracija* prikazuje slika 4.9.

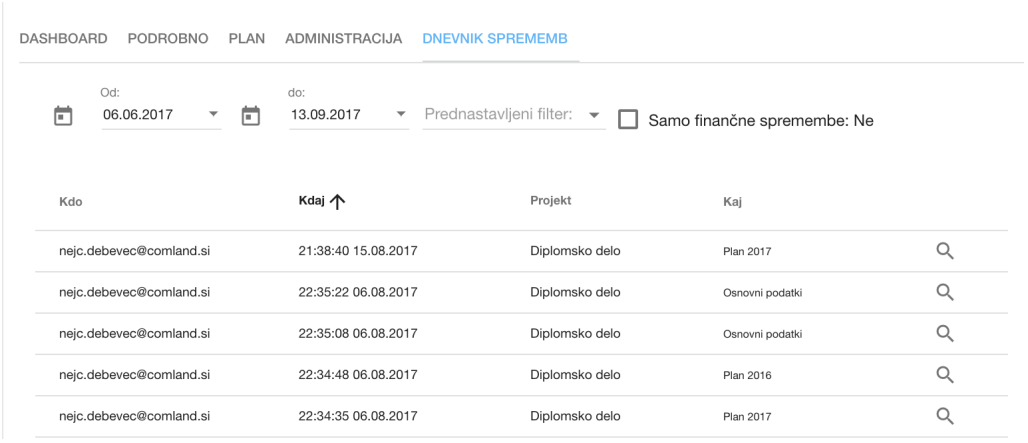



Slika 4.9: Izgled zavihka *Administracija*

Zadnji zavihek je *Dnevnik sprememb*, ki je namenjen prikazovanju vseh sprememb, ki so se zgodile na projektu. V tabeli prikažemo podatke o tem, kdo je naredil spremembe, kdaj in katero sekcijo podatkov je spreminjal (osnovni podatki, podrobni podatki, administracija, finančni podatki). Uporabnikom omogočamo podroben vpogled v posamezne spremembe. V zavihek *Dnevnik sprememb* smo dodali tudi nekaj koristnih filtrov, ki uporabniku izboljšajo preglednost in pohitrijo dostop do želenih podatkov. Implementirali smo dva filtra, ki uporabniku omogočata vnos poljubnih mejnih datumov za prikaz sprememb, ki so se zgodile v tem obdobju. Poleg dveh omenjenih dinamičnih filtrov smo, dodali dva statična, prvi je vnaprej določen časovni filter, ki omogoča prikaz sprememb v zadnjem tednu oziroma mesecu dni. Drugi filter omeji prikaz sprememb samo na te, ki so bile spremenjene iz finančnega vidika. Slika 4.10 prikazuje izgled zavihka *Dnevnik sprememb*.

Za prikaz sprememb smo uporabili JavaScript knjižnico jsondiffpatch, katere funkcija je prepoznava razlik med dvema objektoma. Ta nam je v kombinaciji z verizoniranjem objektov prišla zelo prav in nam je olajšala delo.

Jsondiffpatch smo uporabili tako, da smo pregledali seznam verzij objektov, ker prvi element seznama ni mišljen kot sprememba (objekt ustvarjen), smo pregled začeli od drugega elementa seznama naprej. Vsak trenutni element smo z uporabo jsondiffpatch primerjali s prejšnjim elementom in tako dobili, kakšna je razlika med objektom oziroma kakšne spremembe so bile opravljene. Knjižnica jsondiffpatch implementira funkcijo, ki pripravi objekt,




Slika 4.10: Slika prikazuje izgled zavihka *Dnevnik sprememb*.

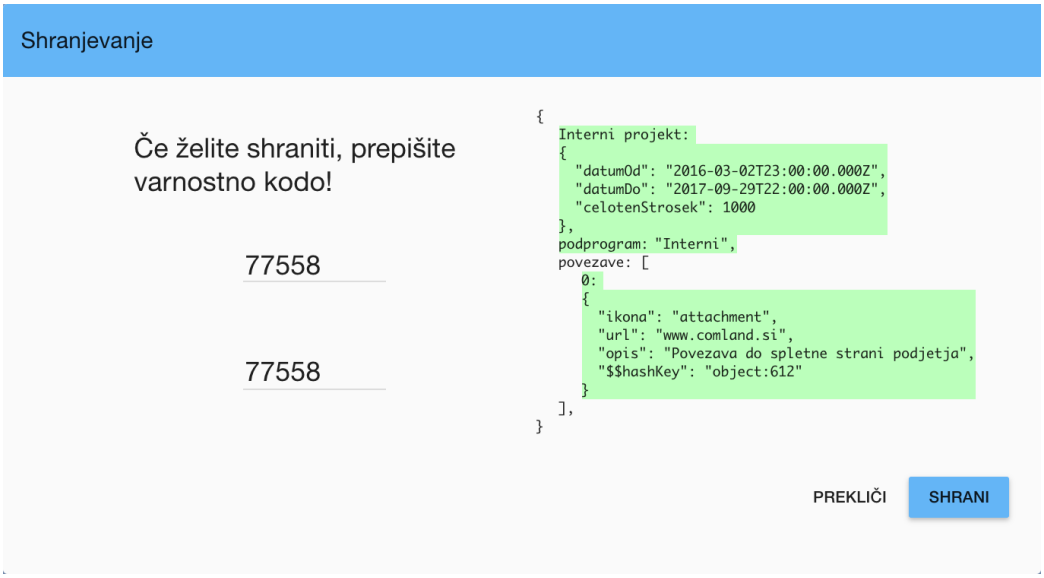
ki vsebuje razliko med dvema objektoma in poskrbi, da se pri prikazu tega objekta, stara verzija obarva v rdečo, nova v zeleno. Tako samo dobljeni objekt samo vključili v HTML dokument. Spremembe so prikazane v JSON objektu. Slika 4.11 prikazuje primer prikaza sprememb na projektu.


Slika 4.11: Izgled pojavnega okna ob prikazu sprememb.

Ob urejanju projekta smo uporabniku omogočili shranjevanje le, ko so se na projektu zgodile spremembe. To smo naredili s kombinacijo AngularJS funkcije $scope.$watch in AngularJS knjižnice angular-input-modified.

Knjižnica je narejena tako, da nadzoruje vse HTML elemente v formi in ob spremembi kateregakoli elementa, nastavi spremenljivko "modified" na pozitivno, katero smo vezali na pogoj za prikaz gumba Shrani. Ker omenjena knjižnica ne podpira nadzora sprememb na seznamih, smo za nadzor dodajanja in brisanja elementov v/iz seznamov (<*md-chips*>), uporabili AngularJS funkcijo $scope.Watch. $Scope.$watch deluje tako, da funkciji podamo argument, ki predstavlja spremenljivko, nad katero želimo nadzorovati spremembe, ta nato na podlagi AngularJS sinhrone povezave objektov iz pogleda v krmilnik in zazna takojšne spremembe.

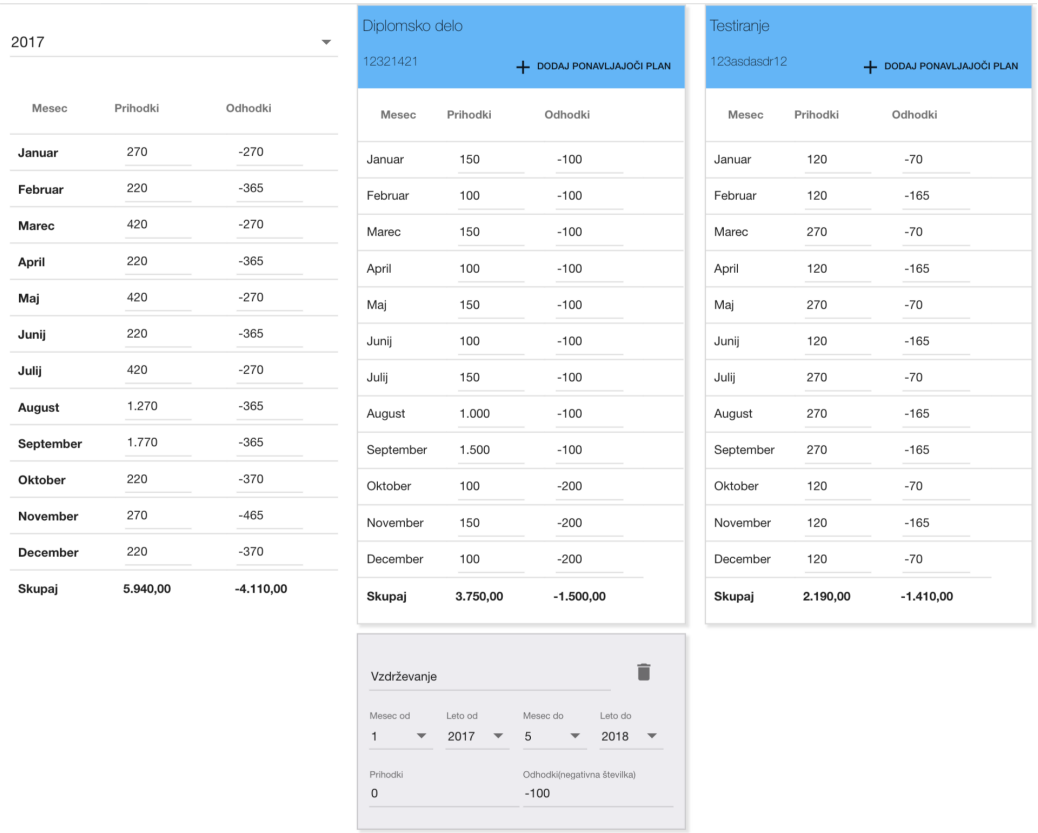
S to funkcionalnostjo smo preprečili nepotrebno shranjevanje istih objektov in tako prihranili v bazi nekaj prostora oziroma časa pri poizvedbah. Pred vsakim shranjevanjem sprememb smo uporabniku prikazali opravljene spremembe, spremembe prikažemo na isti način kot pri zavihku *Dnevnik sprememb*. Vsako shranjevanje mora uporabnik potrditi s prepisom varnostne kode, tako mu preprečimo shranjevanje po pomoti. Slika 4.12 prikazuje izgled pojavnega okna ob shranjevanju.




Slika 4.12: Izgled pojavnega okna ob shranjevanju.

### 4.3.6 Primerjanje/urejanje finančnih planov večih projektov hkrati

Pogled urejanja finančnih podatkov več projektov hkrati vsebuje tri zavihke: *Dashboard, Plan, Dnevnik sprememb*. *Zavihek Dashboard* prikazuje grafe analize finančnih podatkov izbranih projektov. Analize finančnih podatkov in prikaz njenih rezultatov v grafih bomo podrobneje opisali v nadaljevanju. V zavihku *Plan* prikazujemo finančne planirane podatke za vse izbrane projekte. Prikazujemo jih na isti način kot pri pogledu urejanja enega projekta. Slika 4.13 prikazuje izgled pogleda urejanje finančnih podatkov več projektov hkrati.




Slika 4.13: Izgled urejanja finančnih planov večih projektov hkrati

Kot je iz slike razbrano, ponavljajoči plani niso več prikazani v tabeli,

ampak kot posamezni elementi pod projektom. Za takšen prikaz smo se odločili zaradi boljšega pregleda nad tem, kateremu projektu ponavljajoč se plan pripada. Za prikaz planov projektov smo uporabili isto direktivo kot pri urejanju enega projekta. Vsaka tabela projekta ima svojo kontrolo za resetiranje podatkov na vrednosti, ki so bile pred začetkom urejanja. Za to smo uporabili isto knjižnico kot pri urejanju posameznega projekta. Ta omogoča, da pred vsakim urejanjem določimo začetne vrednosti podatkov v formi, na katere se bodo vsakič ponastavili po resetiranju forme.

*Zavihek Dnevnik sprememb*, vsebuje spremembe vseh izbranih projektov. Tabelo sprememb prikažemo s pomočjo direktive "spremembe", ki smo jo pripravili, da na podlagi dobljenih id-jev vseh projektov, dostopa do vseh sprememb izbranih projektov. Dobljene podatke sortira po času spremembe, tako da so najprej prikazani nazadnje posodobljeni podatki. Zavihek vsebuje iste filtre kot zavihek *Dnevnik sprememb* pri posameznih projektih.

Poleg uporabniškega vmesnika smo morali pripraviti tudi programski vmesnik za dostop do podatkov, prikazanih uporabniku. Dostop do podatkov vseh izbranih projektov smo objavili preko URL-ja https://plan-api.comland.si/plani/:id. ID je niz, sestavljen iz vseh ID-jev projektov, ki jih želimo prikazati, ločenih z znakom ','. Ta niz na programskem vmesniku razčlenimo po tem znaku in dobimo ID-je vseh projektov, katerih podatke moramo poslati nazaj. Na podlagi ID-jev naredimo poizvedbe v podatkovno bazo in dobljene podatke tako preuredimo, da vrnemo seznam zadnjih verzij vseh želenih projektov.

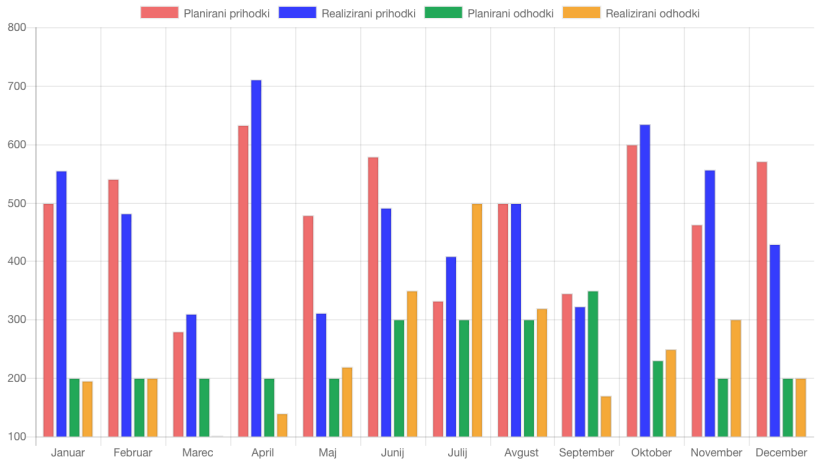
Za dostop do sprememb vseh izbranih projektov smo pripravili API dostop preko URL-ja: https://plan-api.comland.si/Plans/changes/:id. Parameter ID je tu mišljen isto kot pri seznamu zadnjih verzij projektov, torej kot sestavljen niz vseh ID-jev projektov, ločenih z znakom ",". Na podlagi ID-jev smo za vsak projekt posebej naredili poizvedbo in na podlagi verzij projekta, dodajali spremembe v skupen seznam sprememb, ki ga na koncu posredujemo uporabniškemu vmesniku.

### 4.3.7 Implementacija grafov finančne analize

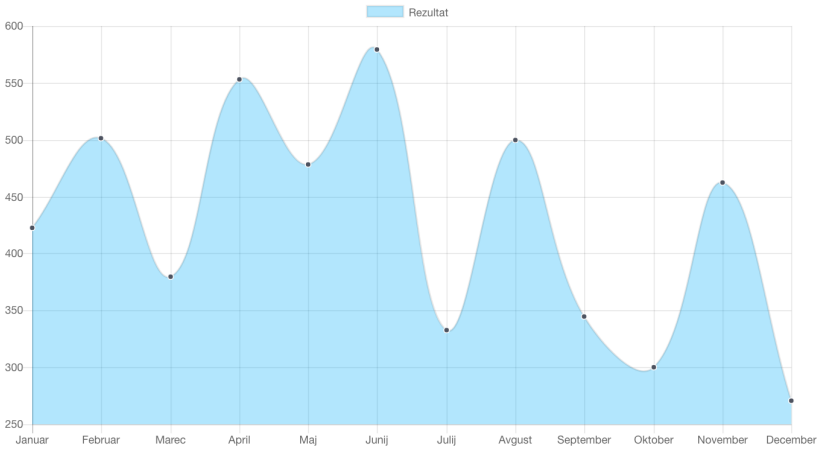
Uporabniku smo za posamezen projekt prikazali dva grafa analize in sicer enega, ki prikazuje rezultat projekta po mesecih in drugega, ki prikazuje primerjavo planiranih in realiziranih prihodkov oz. odhodkov projekta po mesecih. Rezultat prikazuje, čisti dobiček, ki ga dobi podjetje, ko se poračuna vse stroške projekta. Za prikaz rezultata projekta smo uporabili linijski graf, ki omogoča lažjo prepoznavo razlike med meseci. Za prikaz primerjave planiranih in realiziranih finančnih podatkov, smo uporabili stolpični graf, ta omogoča lažje primerjavo podatkov med seboj. Sliki 4.14, 4.15 prikazujeta grafa finančne analize za en projekt.

Pri urejanju oz. pregledu večjega števila projektov hkrati, smo uporabniku prikazali smo en graf finančne analize. Ta graf je za izbrane projekte prikazoval rezultat po mesecih za izbrano leto. Tudi tu smo se odločili za linijski graf, ki za vsak projekt prikazuje linijo, ta pa glede na barvo enolično določa projekt. Slika 4.16 prikazuje graf finančnih rezultatov za večje število projektov.

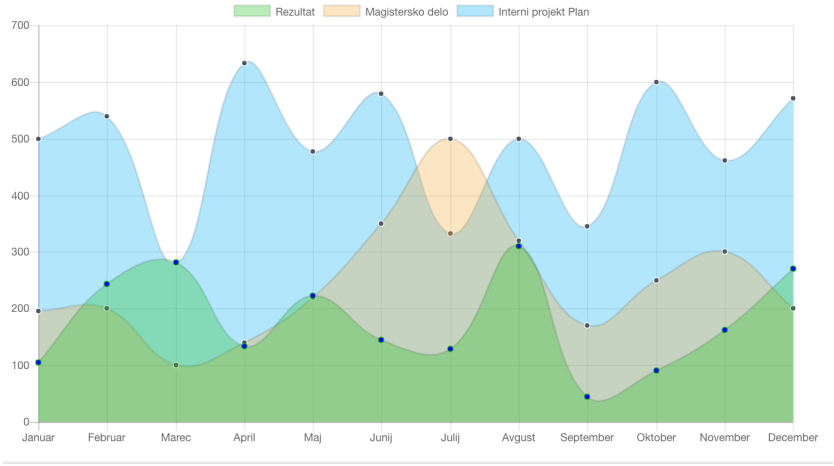
Za pridobitev podatkov finančne analize, smo pripravili API metodo, katera se poveže na podatkovne baze zunanjih sistemov in na njih izvaja poizvedbe po podatkih. Na podlagi interne šifre projetka smo naredili poizvedbo katera nam je za vsak mesec, vrnila podatke o rezultatu in planiranih oz. realiziranih finančnih podatkih na projektu. Dobljene podatke smo morali preurediti tako, da smo iz njih sestavili objekt projekta, kateri je vseboval po letih in mesecih urejene finančne podatke. Te podatke smo pošiljali na odjemlačevo stran, kjer smo iz dobljenih podatkov prikazali grafe.




Slika 4.14: Prikaz primerjave planiranih in realiziranih finančnih podatkov projekta.




Slika 4.15: Prikaz finančnih rezultatov projekta po mesecih za določeno leto.




Slika 4.16: Prikaz finančnih rezultatov večjega števila projektov.

### 4.3.8 Prenos podatkov v druge sisteme

Na strežniku smo nastavili nalogo, ki vsakih 30 minut požene skripto, ki prepiše finančne podatke iz podatkovne baze aplikacije v podatkovno bazo agregiranih sistemov. Tam gredo podatki v obdelavo, za nadaljnjo uporabo. V skripti se povežemo na podatkovno bazo MongoDB, poizvedemo po podatkih vseh projektov. Dobljene podatke smo pripravili in preuredili, da bodo primerni za vstavljanje v podatkovno bazo agregiranih sistemov. S posebno funkcijo smo zgradili dolgo poizvedbo za vstavljanje vseh projektov v podatkovno bazo agregiranih podatkov. To so finančni podatki vseh sistemov, ki jih uporabljajo v podjetju. Grajenje ene velike poizvedbe ni najbolj optimalno, ker lahko nastane problem pri veliki količini projektov, vendar smo se odločili, da zaenkrat izberemo lažjo in hitrejšo verzijo ter da bomo skripto popravili, ko bo potrebno.

## 4.4 Uprabniška izkušnja

Skozi celoten razvoj spletne aplikacije smo poskrbeli za načela dobre uporabniške izkušnje. Skrbno smo izbirali barve komponent, njihovo postavitev

in funkcionalnosti. Poskrbeli smo za preprostost in preglednost

### 4.4.1 Glava aplikacije

V glavo (ang.Header) aplikacije smo dodali nekaj pomembnejših podatkov in funkcij. Dodali smo ikono v obliki hiše (ang.Home), ki uporabniku sporoči, da ga ta gumb preusmeri na domačo oziroma glavno stran, v našem primeru je bila to stran prikaza projektov. Na strani urejanja oziroma vpogleda projekta smo poleg ikone domov, dodali ime projekta, kar uporabniku omogoča, da na hiter način izve, v katerem projektu se nahaja. Na desni strani glave smo dodali ime in priimek uporabnika in pod tem še globalne pravice in pravice nad projektom. Zadnji element glave je bil gumb za odjavo, kateri je uporabnika odjavil in ga preusmeril na prijavno okno. Slika 4.17 prikazuje izgled glave aplikacije.


Slika 4.17: Prikaz izgleda glave aplikacije.

### 4.4.2 Postavitev elementov

Vse gumbe v aplikaciji, ki imajo pomembno akcijo, smo izpostavili tako, da je uporabnik videl, da gre za glavno akcijo. Uporabniku smo omogočili, da lahko nekatere akcije nadzoruje brez miške torej, samo s tipkovnico. Takšen primer je bil pojavno okno ob shranjevanju, saj je uporabniku omogočil takojšen prepis varnostne kode, saj je s klikom na tipko "Enter" sprožil shranjevanje projekta. S pojavnimi okni smo poskrbeli, da se uporabnik ne more zmesti in ve, kaj mora v danem trenutku narediti, saj se je okno pojavilo v ospredju in je zameglilo okolico. Pojavno okno je lahko zaprl le, če je potrdil oziroma preklical akcijo, kliki izven pojavnega okna niso imeli učinka. Gumb shrani je bil vedno postavljen v desnem spodnjem kotu. Za to lokacijo smo se odločili

glede na tok uporabe, ki se iz zgornjega levega kota izvršuje proti desnemu spodnjemu kotu, kar se nam je zdelo najbolj smiselno.

### 4.4.3 Odzvinost aplikacije

Aplikacija je namenjena uporabi na računalniku, vendar ob posebnih priložnostih se bo uporabljala tudi na tabličnih računalnikih in pametnih telefonih. Zato smo aplikacijo zgradili tako, da je odzivna za katerokoli uporabo. Predvsem smo poskrbeli, da aplikacija deluje na različnih brskalnikih za računalnike, predvsem na Google Chrome, Internet Explorer in Mozilla Firefox.

### 4.4.4 Izbira barv

Pri izbiri barv celotne aplikacije smo se osredotočili na enostavne, uporabniku prijazne barve, saj smo s tem izboljšali uporabniško izkušnjo. Za ozadje aplikacije smo zbrali belo barvo, za pisavo črno in s tem poskrbeli, da je vsebina lahko berljiva. Za barvo glave spletne aplikacije smo zbrali svetlo modro, za barvo njene vsebine smo izbrali belo barvo, tako smo dobili dober, ampak ne pretirano vpadljiv kontrast.

## 4.5 Ideje za izboljšave aplikacije

Sama aplikacija je v uporabi že skoraj pol leta. Aplikacija, gledana v celoti, je zelo funkcionalna in ima nekaj zanimivih in uporabnih funkcionalnosti, vendar ima še vedno prostora za izboljšave. Prva izboljšava, ki je smiselna, vendar kar kompleksna, je vmesnik, ki bi generiral poročila za prikaz sprememb. Torej se spremembe ne bi več prikazovale v JSON formatu, ampak kot besedilo, kar bi bilo za uporabnika veliko preprostejše in preglednejše. Ta vmesnik bi lahko dodatno razširili tako, da bi generiral mesečna in letna poročila za projekte in tako bi uporabniku prihranil veliko časa z ročnim ustvarjanjem poročil.

Dodatna funkcionalnost, ki bi uporabniku prinesla dodaten pregled, bi

bilo obveščanje uporabnikov. Torej za vse spremembe, ki se zgodijo na projektu, nad katerim ima uporabnik pravice, bi bil ta obveščen, kdaj so se zgodila in prikazan bi mu bil predogled sprememb. Prav tako bi bil obveščen, ko bi mu bile dodeljene nove pravice nad nekim projektom.

Smiselno bi bilo dodati še kakšne grafe analize podatkov, ki bi uporabniku izboljšali nadzor nad uspešnostjo projekta.

# Poglavje 5

# Uporaba metoda pri razvoju aplikacije

Razvoj spletne aplikacije smo se lotili po metodi Scrum. Kako smo razdelili vloge, načrtovali iteracije in se splošno vključevali v proces metodologije Scrum, bomo predstavili v spodnjih podpoglavjih.

## 5.1 Vloge

Pri procesu razvoja po metodi Scrum so ključnega pomena vloge. Zato smo najprej določili vloge.

### 5.1.1 Skrbnik izdelka

Vloga skrbnika izdelka je bila dodeljena direktorju tehnološkega razvoja. Za to vlogo je bil zadolžen, ker ima največ znanja glede vsebine spletne aplikacije. Ves čas je skrbel, da je bil seznam zahtev, posodobljen glede na zahteve, ki so bile potrebne za implementacijo. Vedno je bil na voljo razvojni skupini za dodatna vprašanja oziroma razlago zahteve. Skrbel je za dobro vzdušje v razvojni skupini in držal motivacijo razvijalcev na visoki ravni.

### 5.1.2 Skrbnik metodolgije

Vlogo skrbnika metodologije je prevzela, izkušena programerka, ki je osredotočena na razvijanje uporabniških vmesnikov spletnih aplikacij. Ima že veliko izkušenj z razvijanjem po procesu metodologije Scrum, vendar še nikoli ni bila v vlogi skrbnika metodologije. Da ni prihajalo do težav pri razvoju po procesu Scrum, je vsakega člana razvojne skupine seznanila z načinom dela pometodologiji Scrum. Vedno je skrbela, da smo vsi v razvojni skupini, točno vedeli, kaj uporabniška zgodba zahteve od nas in ni prišlo do napak pri implementaciji.

### 5.1.3 Razvojna skupina

Razvojna skupina je bila sestavljena iz dveh mladih razvijalcev. Noben od njiju ni imel prejšnjih izkušenj iz metodologije Srum, zato je bila velika odgovornost na skrbniku metodologije, da ju pravilno in efektivno vpelje v proces razvoja po metodologiji Scrum. Eden izmed razvijalcev je prvič sodeloval pri razvoju spletne aplikacije. Njegovo znanje izbranih tehnologij na projektu je bilo začetno, skoraj neobstoječe. Zato je bil pritisk na skrbniku metodologije toliko večji, saj je moral dokaj realno oceniti sposobnosti razvijalcev in zagotavljat konstantno, ne prepočasno hitrost razvoja. Drugi razvijalec je bil mladi razvijalec, ki je imel nekaj izkušenj z razvojem spletnih aplikacij, predvsem z zalednimi sistemi. Zaradi potreb na drugih projektih, ta razvijalec ni na projekt sodeloval celoten delovnik. Zaradi potreb na drugih projektih, se je razvojni skupini pridruževal dinamično glede na potrebe. Včasih je razvoju priskočila na pomoč skrbnica metodologije, ki je s svojim znanjem prinesla dodatno vrednost razvojni skupini. To, da se je skrbnika metodologije dodalo kot razvijalca, ni bilo najbolj optimalno, vendar zaradi pomanjkanja delovne sile, ni bilo druge izbire.

## 5.2 Ocenjevanje zahtevnosti uporabniške zgodb

Za izdelavo načrta izdaje je potrebno, da so vse uporabniške zgodbe iz seznama zahteve ocenjene. Ker sta bila člana razvojne skupine neizkušena, so oceno uporabniških zgodb iz seznama zahtev na podlagi svojih dolgoletnih izkušenj v razvoju spletnih aplikacij, določila skrbnico metodologije in skrbnik izdelka. Poleg ocene zahtevnosti je skrbnik izdelka prioritiziral seznam zahtev. Tako smo lahko na podlagi ocen in prioritiziranega seznama pripravili načrt izdaje. Z načrtom izdaje smo določili okvirno vsebino vsake iteracije.

## 5.3 Orodja za vodenje projetka

Za vodenje poteka razvoja projekta smo uporabili dve orodji, in sicer spletno aplikacijo FogBugz in navadno tablo. V FogBugz aplikaciji smo beležili porabljene ure pri implementaciji posameznih zahtev in nasploh v iteraciji oziroma celem projektu. Tablo smo uporabljali za prikaz trenutnega dela. Na tabli je bila narisana tabela, ki je imela pet stolpcev: story, to do, in progress, in testing, done. Vsako uporabniško zgodbo iz seznama zahtev smo napisali na listek in ga dodali na tablo v stolpec story. Vse trenutne zahteve iteracije smo premestili v stolpec to do, vsaka zahteva, ki je bila v procesu implementacije, smo jo postavili v stolpec in progess. Ko smo končali z implementacijo le-te, smo jo premaknili v stolpec In testing. Če je zahteva uspešno prestala teste, smo jo premaknili v stolpec Done, če ne, smo jo postavili nazaj v In progress. .

## 5.4 Potek iteracije

Ker razvojna skupina ni imela izkušenj z uporabo metodologije Scrum, smo dolžino iteracije določili samo na podlagi izkušenj skrbnika izdelka in skrbnika metodologije. Skrbnik izdelka in skrbnica metodologije sta se odločila za dvotedenske iteracije. Vsaka iteracija se je začela z načrtovanjem iteracije

(ang. Sprint planing) in zaključila s pregledom končane iteracije (ang. Sprint Review Meeting) in oceno končane iteracije (ang. Sprint Retrospective meeting), omenjeni sestanki so se odvijali ob ponedeljkih na 2 tedna. Poleg omenjenih sestankov se je vsak dan ob 9. uri odvijal dnevni sestanek, ki je trajal 15 minut.

Kot smo omenili, se je iteracija začela z načrtovanjem iteracije, ki je potekalo ob ponedeljkih vsaka dva tedna. Takrat smo se v sejni sobi zbrali vsi sodelujoči. Sestanek je bil razdeljen na dva dela, v prvem delu je skrbnik izdelka predstavil uporabniške zgodbe, ki so bile definirane v seznamu zahtev. Tako smo vsi člani izvedeli dovolj informacij, da smo lahko kasneje te uporabniške zgodbe ocenili. V drugem delu sestanka je sledilo ocenjevanje posameznih uporabniških zgodb in nato dodelitev le-teh posameznim razvijalcem. Ocenjevanja smo se lotili po metodi poker planiranja. Ker v razvojni ekipi nismo točno vedeli, kaj poker planiranje je, smo posvetili krajši del sestanka predstavitvi metode poker planiranja. Ko smo osvojili način ocenjevanja po metodi poker planiranja, smo se lotili ocenjevanja po seznamu zahtev. Zaradi pomanjkanja izkušenj z razvojem spletnih aplikacij so na začetku ocene razvojne skupine odstopale od realnih, da se nismo preveč oddaljili od realnih ocen, je poskrbela skrbnica metodologije. Ko je videla, da je ocena precej odstopala od realne, nam je zahtevo še enkrat podrobneje razložila in če še to ni bilo dovolj, se je lotila podrobne razlage iz tehničnega vidika, dokler nam ni bila slika jasna.

Skrbnica metodologije je manjša odstopanja pustila, saj je želela, da se na napakah učimo in dobimo potrebne izkušnje. To nam je dalo vedeti, kako lahko napačno ocenjen seznam zahtev močno vpliva na izvedbo posamezne iteracije oziroma na razvoj izdaje.

Glede na to, da je bil eden izmed razvijalcev brez izkušenj v programiranju spletnih tehnologij, se je na začetku njemu dodeljevalo osnovne naloge, s katerim je dobil osnove spletnih tehnologij in strukturo projekta. Tako smo ga lahko postopoma vpeljali do razvoja kompleksnih zahtev in njegove ne izkušnje niso imele drastičnega vpliva na izvedbo iteracije.

## 5.5 Pravila za koncept zaključene zahteve

Postavili smo tudi pravila za koncept zaključene zahteve in se tako izognili nepotrebnim objavam funkcionalnosti z napakami v produkcijskem okolju in ponovnemu odpiranju že zaključenih zahtev. Da smo lahko zaključili zahtevo, je morala ta prestati naslednja pravila:

- komentarji kode,
- strukturirana koda ,
- implementirana zahteve pretestirana s strani skrbnika izdelka in skrbnice metodologije,
- napisana dokumentacija, ki razlaga funkcionalnost implementirane zahteve,
- konsistenten izgled (uporaba isith oblikovalnih prvin skozi celotno aplikacijo).

## 5.6 Razvoj po iteracijah

Na podlagi neizkušen razvojne skupine z razvojem spletnih tehnologij in z metodologijo Scrum, smo se na začetku odločili za manjšo hitrost razvoja. Prva iteracija je vsebovala seznanitev s strukturo projekta. Tako smo začetno hitrost razvoja določili na 110 ur, v katero je vštet član s polnim delovnikom in pa član s približno polovičnim delovnikom(dinamično glede na potrebe razvoja pri tem oziroma drugih projektih v podjetju).

### 5.6.1 Prva iteracija

Ne glede na to, da so člani razvojne skupine porabili že kar nekaj ur za spoznavo z okoljem in strukturo projekta, smo načrtovanju iteracije namenili kar nekaj časa. Pri ocenjevanju zahtev smo pri vsaki oceni dodali nekakšen rezervni dodatek, ki je bil mišljen za testiranje in popravke, saj so

bili pričakovani glede na izkušnje razvijalcev. Spodnji seznam zahtev iteracije vsebuje uporabniško zgodbo in zraven njeno oceno za implementacijo v urah. Kot smo omenili, smo začetno hitrost razvoja določili na 110 ur. Na podlagi rezultatov prve iteracije bomo lahko nato hitrost razvoja realneje ocenili.

Seznam zahtev prve iteracije:

- Prijava v aplikacijo: 20 ur.
- Načrtovanje osnovnega izgleda aplikacije: 5 ur.
- Prenos projektov iz drugih sistemov: 13 ur.
- Prikaz vseh projektov: 20 ur.
- Dodajanje projekta(Osnovni podatki, podrobni podatki): 40 ur.
- Sortiranje projektov v tabeli po stolpcih: 2 uri.
- Prikaz samo odprtih projektov: 3 ure.
- Iskanje med projekti: 8 ur.

Čeprav je bila to prva iteracija za razvojno skupino, je iteracija potekala normalno in nemoteno. Vse zahteve iz seznama zahtev se niso implementirale, saj kot smo omenili, smo veliko časa porabili za spoznavo strukture projekta in vanj vključenih tehnologij in nismo mogli realno oceniti hitrosti razvoja zaradi neizkušenih razvijalcev. Iz zgornjega seznama so bile implementi rane vse zahteve razen dodajanja projekta. V prvih dveh tednih razvoja je skrbnik metodologije imel veliko dela z odgovarjanjem na vprašanja razvojne skupine, vendar so se ti hitro učili. Veliko problemov je bilo z dvema praviloma v konceptu zaključka zahteve, in sicer s komentarji in strukturo kode.

### 5.6.2 Druga iteracija

Za drugo iteracijo smo hitrost razvoja malenkost znižali na 88 ur. Zahteva dodajanje projekta, je bila v prvi iteraciji že delno razvita, zato smo jo še

enkrat ocenili glede na to, koliko je še potrebnega dela za njeno dokončno implementacijo. Seznam zahtev za drugo iteracijo so sestavljale naslednje zahteve:

- Dodajanje projekta(Osnovni podatki, podrobni podatki): 20 ur.
- Urejanje projekta(Osnovni podatki, podrobni podatki):13 ur.
- Izbiranje večjega števila projektov: 2 uri.
- Urejanje večjega števila projektov: 40 ur.
- kontrola nadzora shranjevanje uporabnika: 13 ur.

### 5.6.3 Tretja iteracija

Hitrost razvoja v tretji iteraciji smo na podlagi dejstva, da sta bila oba razvijalca polni delovnik na projektu, povišali na 128 ur. Tretja iteracija je potekala nemoteno, napake iz prejšnjih so se zmanjševale, vendar so bile še vedno prisotne. Odločili smo se, da v tej iteraciji dodamo sestanek pregleda kode, kjer se celotna razvojna skupina usede in pregleda kodo implementirane zahteve. Tako smo pregledali, če je možno kje, kakšno kodo optimizirati oziroma najti boljšo rešitev. V tej iteraciji je skrbnica metodologije pretestirala uporabniško izkušnjo in našla vse napake v zvezi z oblikovanjem in postavitvijo strani.

Delno se je tudi sama postavila v vlogo razvijalca, pomagala je pri postavitvi logike za zahtevo dnevnik sprememb na projektu. Na pregledu iteracije sta se nam pridružila direktor in lastnik podjetja, oba sta bila z videnim zadovoljna.

Spodnji seznam prikazuje seznam zahtev tretje iteracije:

- Administracija nad projekti: 40 ur.
- Administracija nad aplikacijo: 20 ur.
- Testiranje uporabniške izkušnje(oblikovanje,postavitve): 8 ur.

- Dodajanje ponavljajočih finančnih planov: 13 ur.
- Dnevnik sprememb na projektu: 20 ur.
- Dnevnik sprememb na večih projektih: 13 ur.

Iteracija je bila uspešna, implementirane so bile vse zahteve. Dobili smo tudi že nekaj odzivov in prijav napak s produkcijskega okolja, kjer uporabniki testirajo aplikacijo. Prijavljene napake smo vključili v seznam zahtev.

### 5.6.4 Četrta iteracija

Četrta iteracija je bila zadnja iteracija v razvoju aplikacije. V tej iteraciji smo dodali grafe analize, ki prikazujejo planirane finančne podatke in realizirane finančne podatke. Ker smo dobili že veliko prijav glede manjših napak v aplikaciji, smo namenili nekaj časa tudi popravkom le-teh. Hitrost razvoja je bila v tej iteraciji postavljena na 106 ur.

- Popravki prijavljenih napak: 13 ur.
- Scripta za prenos podatkov v druge sisteme: 13 ur.
- Prikaz grafov analize uspešnosti enega projekta: 40 ur.
- Prikaz grafov analize uspešnosti večih projektov: 40 ur.

Iteracija se je končala uspešno, vse zahteve so bile implementirane, prijavljene napake so bile odpravljene. Z zaključkom te iteracije se je projekt zaključil. Kot zadnje dejanje smo imeli pregled in oceno poteka razvoja celotnega projekta. Ugotovitve in analizo uporabo metodologije Scrum smo opisali v spodnjem podpoglavju.

## 5.7 Analiza uporabe Scrum-a pri razvoju aplikacije

Razvoj projekta je bil v celoti uspešen, tako iz funckionalnostnega kot iz časovnega vidika. Razvoj aplikacije in pri tem uporaba procesa Scrum, je bila namenjena učenju mladih neizkušenih programerjev, da dobijo občutek glede poteka razvoj projekta in kako se odraža uporaba procesa Scrum pri razvoju.

### 5.7.1 Analiza vlog v procesu Scrum

Ker smo začeli proces metodologije Scrum z razdelitvijo vlog, bomo najprej namenili nekaj besed glede vlog in njihovih dobrih oziroma slabih lastnosti v našem procesu.

#### Skrbnik izdelka

Skrbnik izdelka je opravil svoje delo odlično, tu so bile vidne njegove dolgoletne izkušnje iz razvoja različnih projektov po metodologiji Scrum. Razvojna skupina in skrbnica metodologije so se lahko vedno obrnili nanj s kakršnimi koli vprašanji glede projekta. S svojimi komunikacijskimi spretnostmi je motiviral razvijalce in skrbel, da je bilo vzdušje v razvojni skupini na visoki ravni.

#### Skrbnica metodologije

Glede na to, da skrbnica metodologije, ni imela izkušenj v tej vlogi, je pomanjkanje le-teh nadomestila s svojim dobrim teoretičnem znanjem o metodologiji Scrum. Svoje delo je opravljala dobro in učinkovito. Razvojni skupini je nenehno vlivala nova znanja glede metodologije in skrbela, da so ne glede na neizkušnje hitro vključili v proces Scrum.

Zelo dobro je razvojni skupini razlagala uporabniške zgodbe in s tem zagotovila, da določene zahteve ne bodo zaradi nejasnosti narobe implementirane.

Velik plus pri celotnem procesu Scrum, je bil dober odnos med skrbnico metodologije in skrbnikom izdelka. Zelo dobro sta se usklajevala in skrbela, da na razvojno skupino ne vplivajo zunanje motnje. Seznam zahtev je bil vedno lepo urejen in dodelan, opisi uporabniških zgodb so bili na visoki ravni, s tem je bilo poskrbljeno, da so razvijalci lažje razumeli zahteve.

**Razvojna skupina**

Še največ problemov pri procesu Scrum je povzročala razvojna skupina. Zaradi neizkušenosti razvijalcev so se pojavljali problemi pri ocenah zahtev, le te pa so vplivale na samo izvedbo posameznih iteracij. Z vsako iteracijo so se tudi ocene bližale realnosti. Tako, smo že po prvi iteracij ocenili približno sposobnost razvojne skupine in na podlagi te ocene, nato načrtovali iteracije. Pri razvojni skupini je bila slaba praksa to, da člani niso imeli dobrih izkušenj iz tehnologij vključenih na projektu. To bi lahko popravili s tem da bi na projekt vključili dodatne izkušene razvijalce, ki bi s svojimi izkušnjami pomagali mladim razvijalcem. Vendar sam projekt je bil namenjen učenju in uvajanju novih razvijalcev v metodologijo in sam razvoj projektov. Vsi omenjeni problemi razvojne skupine so povezani z izkušnjami, torej glede na izkušnje razvijalcev pričakovani, vendar je razvojna skupina neglede na vse napke, imela lasnosti, ki je izmed pomembnejših in to je delavnost. Člani razvojne skupine so pokazali veliko željo po učenju, dokazovanju in pridobivanju novih izkušenj, to je olajšalo sam razvoj. S samoiniciativnostjo in hitrim razumevanjem, so hitro popravljali omenjene napake.

Velik minus razvojne skupine in samega proces razvoja je bil ta, da je bil v razvojni skupini samo en član, ki je sodeloval na projektu poln delovnik. Metodologija priporoča, naj bi bili v razvojni skupini vsaj trije razvijalci, res da, je bil projekt namenjen učenju, vendar bi z razvojno skupino vsaj treh razvijalcev lahko lažje in boljše izvajali sam proces. Omogočilo bi sprostitev pritiska, ki se je pojavil na članih skupine, kar bi odražalo tudi na boljšem učenju samega procesa.

Razvijalci so imeli veliko problemov s konceptom zaključene zahteve, saj

se niso dobro držali testnih pravil. Komentarij kode niso bilo primerni, njihova vsebina ni najbolje opisovala vsebine kode, poleg veliko pravopisnih napak in "domačih" izrazov, je bila sama struktura komentarjev napačna. Pri strukturi kode se je pojavljalo to, da so v kodi ostajali razni komentarji in odvečne "testne" vrstice. Komponente kode niso bile med seboj lepo poravnane, kar je zelo vplivalo na preglednost kode. Skrbnica metodologije je nenehano opozarjala razvojno skupino glede omenjenih napak, vendar so se le te zaradi njihove površnosti počasi popravljale. To je vplivalo na sam razvoj projekta, saj so morali razvijalci porabiti dodaten čas za popravke teh napak.

### 5.7.2 Analiza sestankov

Pri izvedbah sestankov ni bilo kritičnih napak, tu se je poznal širok spekter znanja skrbnice metodologije o metodologiji Scrum. Poleg omenjenih težav pri načrtovanju iteracij, zaradi nerealnih ocen hitrosti razvoja, je načrtovanje potekalo skladno z načeli. Mogoče se je na začetku poznalo rahlo nesodelovanje razvojne skupine na sestankih, vendar smo to pripisali začetni tremi oz. sramežljivosti. Težava se je pojavljala pri izvedbah dnevnih sestankov, tam smo na začetku večino sestankov zavlekli krepko čez priporočenih 15 minut. To se je dogajalo predvsem zato, ker so se na sestankih začeli reševati probleme. Scrum strmi k temu, da na dnevnih sestankih problemov ne rešujemo, jih le omenimo, torej kje se nam je pri razvoju ustavilo oz. s kakšnimi težavami smo se srečevali pri implementacij določenih zahtev. Tu gre krivda na skrbnico metodologije, ki je pustila, daje sestanek zašel s priporočenih tematik. Ko je to opazila, bi morala temo prekiniti in razvijalcu, ki je imel problem pri razvoju, predlagati da se z njo oz. drugim razvijalcem vsedeta po končanem sestanku in s skupinimi močmi najdeta rešitev.

Vsi sodulujoči na projektu smo se strinjali, da je dodatno vrednosti pri procesu razvoja prinesla metoda poker planiranja. Z metodo smo realno ocenili izvedbe posameznih uporabniških zgodb. Mladim razvijalcem smo omogočili, da dobijo občutek za ocenjevanje zahtev, kar jim bo koristilo pri

nadaljnih razvojih. Na sestankih načrtovanja iteracije smo s poker planiranjem desegli red in disciplino pri samem ocenjevanju. Ni bilo nepotrebnih skakanj v besedo in usklajevanj kdo naj prvi ocenjuje.

Po vsaki končani iteracij smo izvedli pregled iteracije, tu so sodelovali vsi, ki so želeli videti napredek pri razvoju projekta v končani iteracij. Sam pregled je potekal neorganizirano in ni se točno vedelo kdo ga vodi. Tu je prišlo da slabih predstavitev implementiranih zahtev, saj niso imeli vsi člani razvojne skupine dobrih predstavitvenih vrlin. Temu bi se lahko izognili tako, da bi določili vodjo, ki bi vodil pregled in s tem poskrbeli za konsistentno in razumljivo predstavitev.

Ocen iteracij nismo izvajali, saj smo bili mnenja, da smo napake opažali že med samim razvojem in jih poskušali odpraviti. Res da smo določene napake opazili med samim izvanjanjem iteracij in jih tudi odpravili, vendar sestanek za oceno iteracije ni namenjen samo iskanju slabih stvari, ampak tudi dobrih. Z izvajanem ocen iteracij, bi dobili boljši pregled nad napakami, ki se dogajajo med izvajanjem procesa in omogočili diskusijo in iskanje rešitev za njih. S pregledom pozitivnih stvari, ki so se dogajale med procesom Scrum, bi pa razvojni skupini vlili dodatno motivacijo.

### 5.7.3 Analiza procesa Scrum v celoti

Med samim razvojem smo opazili veliko prednosti metodologije Scrum. Predvsem prilagajanje razvoja dinamičnemu dodajanu novih zahtev. To pomeni da ni potrebe po popolnem končnemu seznamu zahtev že na začetku projekta, vendar se lahko zahteve po potrebi doda med razvojem. To je bilo v našem primeru zelo koristno, saj se ni točno vedelo na začetku projekta kako se bo odvil. Nekaj večjih funkcionalnosti je bilo dodanih naknadno. Z metodo Scrum smo dobili dobro organizacijo pri razvoju projekta, točno se je vedelo kdo ima kakšno nalogo in do kdaj jo mora naredit. Za organizacijo znotraj razvojne skupine, skrbi razvojna skupina sama, kar omogoča razvijalcu, da si sam izbere nalogo, ki mu je najbolj pisana na kožo in s tem izboljša produktivnosti. Vsak razvijalec si izbere toliko nalog kolikor sam misli da

jih je sposoben opraviti v posamezni iteraciji, kar vodi k boljši iskoriščenosti časa in zaradi ne natrpanega delovnika, k bolj sproščenem delovnem vzdušju znotraj skupine.

Razvojna skupina je bila vedno seznanjena z vsem dogajanjem pri razvoju, kar je prednost, saj se točno ve kdo kaj dela, kaj je naredil, kaj bo delal, kakšne težave ima. Ravno zato so dnevni sestanki tako pomembni, saj je celotna razvojna skupina seznanjena o dogajanju pri drugih članih.

Samo izvajanje procesa Scrum ni bilo težko, saj je celotno podjetje že vpeljano v agilen razvoj projektov, kar je bila velika prednost. Za kakršne koli nasvete oz. pomoč smo se lahko obrnili na kogarkoli v podjetju.

Celotno gledalo je bil razvoj aplikacije po metodi Scrum uspešen. Neglede na težave s katerimi smo se srečevali in neizukšnjami razvojne skupine je bila aplikacija uspešno implementirana in je že v uporabi s strani uproabnikov. Mladi neizkušeni razvijalci so dobili še kako potrebne izkušnje tako iz programiranja kot iz metodologije Scrum.

# Poglavje 6

# Sklepne ugotovitve

V tem diplomskem delu smo se posvetili razvoju preproste, odzivne, uporabniku prijazne spletne aplikacije. Ravoj je potekal po metodologiji Scrum. Uporaba Scruma je bila namenjena uvajanju v agilen razvoj projektov, katero smo kasneje uporabljali v podjetju. Sama spletna aplikacija v osnovi ni kompleksno strukturirana, vendar ima veliko funkcionalnosti. Z manjšimi dodatnimi funckionalnostmi smo uporabniku izboljšali uporabniško izkušnjo. Da uporabnik ne bo imel problemov pri sami uporabi aplikacije, smo strmeli k jasni predstavi podatkov oz funkcionalnosti posameznih delov aplikacije. Aplikacija je trenutno uporabljena znotraj podjetja in odzivi uporabnikov so bili pozitivni. Izpostavili so predvsem preprostost, dinamičnost in hitrost aplikacije. Razvoj po metodi Srcum je potekal po načrtih, zaradi neizkušenosti v tehnologijah na projektih, smo imeli nekaj težav pri ocenah zahtev, kar nam je pri prvi iteraciji povzročalo kar velike probleme, vendar smo to kasneje v iteracijah rešili. Nekaj problemov se je pojavilo tudi pri sestankih, vendar vse omenjene težava lahko pripišemo neizkušenosti. Vsi člani skupine so pokazali veliko pozitivnih lastnosti, predvsem delavnost, samoiniciativnost ter hitro učenje. Vzdušje je bilo skozi celoten cikel razvoja na zelo visokem nivoju, bilo je sproščeno, zabavno in delavno, kar je prava kombinacija za uspešen razvoj projekta.

Med izdelavo diplomske naloge smo se naučili ogromno novih in koristnih

stvari. Dobili smo izkušnje iz vseh omenjenih tehnologij na projektu. Glede na to, da nismo imeli veliko izkušenj iz izdelave spletnih tehnologij, smo izboljšali tudi razumevanje delovanja le teh oz. spleta nasploh. Drastično se je zvišalo razumevanje vodenje in razvoja projektov, predvsem pa metodologije Scrum. Dobili smo občutek skupinskega dela, kar nam bo pri nadaljnih razvojih prišlo zelo prav.

# Literatura


[1] Angularjs. Dosegljivo: https://en.wikipedia.org/wiki/AngularJS, 2017. [Dostopano: 23. 07. 2017].

[2] Css. Dosegljivo: https://sl.wikipedia.org/wiki/CSS, 2017. [Dostopano: 22. 07. 2017].

[3] Expressjs. Dosegljivo: https://expressjs.com/, 2017. [Dostopano: 23. 07. 2017].

[4] Git. Dosegljivo: https://sl.wikipedia.org/wiki/Git, 2017. [Dostopano: 23. 07. 2017].

[5] Html. Dosegljivo: http://mrvar.fdv.uni-lj.si/sola/VIS2/html/htmlslo.html, 2017. [Dostopano: 22. 07. 2017].

[6] Html wikipedia. Dosegljivo: https://sl.wikipedia.org/wiki/HTML, 2017. [Dostopano: 22. 07. 2017].

[7] Javascript. Dosegljivo: https://sl.wikipedia.org/wiki/JavaScript, 2017. [Dostopano: 23. 07. 2017].

[8] Mongodb. Dosegljivo: https://www.mongodb.com/, 2017. [Dostopano: 18. 07. 2017].

[9] Mongodb wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/MongoDB, 2017. [Dostopano: 18. 07. 2017].

[10] Nodejs. Dosegljivo: https://nodejs.org/en/about/, 2017. [Dostopano: 18. 07. 2017].

[11] Nodejs wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/Node.js, 2017. [Dostopano: 18. 07. 2017].

[12] Scrum. Dosegljivo: https://www.mountaingoatsoftware.com/agile/scrum, 2017. [Dostopano: 19. 07. 2017].

[13] Scrum wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/Scrum_(software_development), 2017. [Dostopano: 18. 07. 2017].

[14] Slack slika. Dosegljivo: https://get.slack.help/hc/en-us/article_attachments/115000754027/Group_2x.png, 2017. [Dostopano: 18. 07. 2017].

[15] Slika diagrama poteka v iteraciji. Dosegljivo: http://scrum-basics.blogspot.si/2014/12/release-planning-in-scrum.html, 2017. [Dostopano: 01. 08. 2017].

[16] Slika diagrama poteka v izdaji. Dosegljivo: http://scrum-basics.blogspot.si/2014/12/release-planning-in-scrum.html, 2017. [Dostopano: 01. 08. 2017].

[17] Adam Bretz Colin J Ihrig. *Full Stack JavaScript Development With MEAN: MongoDB, Express, AngularJS, and Node.JS*. SitePoint, 2014.

[18] Jeff Dickey. *Write Modern Web Apps with the MEAN Stack: Mongo, Express, AngularJS, and Node.js*. Peachpit Press, 2014.

[19] Ken Schwaber. Agile project management with scrum. pages 1–14, 132–143, 2004.